Studying the Performance of Cooperative Delivery Techniques to Support Video-on-Demand Service in IPTV Networks

Aytac Azgin, *Member, IEEE,*, Ghassan AlRegib, *Senior Member, IEEE,* Yucel Altunbasak, *Fellow, IEEE*
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, Georgia 30332–0250
Email: aytaca@gatech.edu, alregib@gatech.edu, yucel@ece.gatech.edu

***Abstract*—In this paper, we study the use of peer-assisted server-based cooperative transmission strategies in IPTV networks for the delivery of on-demand services to end users. The proposed techniques aim to support the resource efficient delivery of on-demand content to end users in a timely manner. Within the proposed framework, a cooperative transmission strategy suggests that users who have access to the requested content cooperatively transmit to the targeted set of users. In doing so, we can minimize the servicing requirements at the server side and improve the scalability performance in the network. We conducted extensive simulations using different request arrival models and showed that significant performance improvements can be achieved with the proposed delivery techniques to enable efficient access to an ever growing on-demand content. We also showed the robustness of the proposed techniques in regard to variations observed in network state.**

***Index Terms*—Internet Protocol TeleVision, Video-on-Demand service, cooperative networking, peer-to-peer streaming, content delivery networks.**

I. Introduction

In recent years, user demand for the delivery of broadcast content over IP infrastructure has seen a significant increase [1]. Through rapidly evolving server-assisted on-demand content delivery services, such as Youtube and Netflix [2], live broadcast services (also known as the IPTV services), such as U-verse, and peer-assisted on-demand streaming services, such as Sopcast or PPLive [3], [4], we have observed a substantial increase in the amount of content that is immediately made available to the end users [5], [6]. Due to this mutually evolving content delivery cycle, the amount of bandwidth required to timely service the user requests has also seen a significant increase [7], [8]. To continue to support the diverse service quality requirements for the continually increasing user demand, it is of critical importance to come up with resource efficient content delivery techniques.

An important characteristic of content delivery networks (live or on-demand) is that a substantial portion of the video traffic (or the network load) can be attributed to the most popular content [9], [10], [11], [12]. That is because a linear increase in the popularity of a content results in an exponential increase in the number of users receiving the content, thereby leading to popularity models based on the power law distribution (*e.g.*, Zipf distribution [12]). To overcome the sudden increase in user demand, caching policies are typically utilized to distribute the load in the network by giving higher priority to the delivery of more popular content (see [13] and the references within). However, continual increase in network size and user demand may drive the cost of caching policies higher than what is initially expected by the service providers, thereby limiting their efficiency in handling the on-demand traffic. The bandwidth requirements for the received requests (*e.g.*, 3 $Mbps$ for the standard-definition (SD) streams and 8 $Mbps$ for the high-definition (HD) streams) and the–likely to be long–activity duration associated with the content requested therefore make it a necessity to more efficiently manage the available resources by further distributing the network load.

To achieve these objectives, we propose a cooperative delivery framework that requires peers, who have immediate access to the content being requested, to transmit together to maximize the resource usage efficiency in the network. The origin of our idea lies in the *cooperative diversity* principle utilized in wireless networks to maximize the network lifetime (see [14] and references within). Cooperative diversity creates a virtual multiple-input-multiple-output (MIMO) array in the network to efficiently distribute the energy resources when delivering traffic from one node to another through multiple relay points. Through successive cooperative transmissions, energy utilization per node can be significantly reduced at each transmission hop, thereby allowing the nodes to utilize their energy resources more conservatively and evenly to achieve a higher network lifetime.

However, cooperation in wireless networks is limited to the use of individual nodes, hence the performance improvements strongly depend on the network topology. We do not have such restrictions in wireline networks due to the use of dedicated servers that can actively participate in the cooperation phase to support the delivery process. Therefore, we can utilize the results from cooperative networks in a more flexible manner. Specifically, in a wireline setting, to achieve similar performance improvements, we can replace node budgets that are correlated to energy usage in a wireless setting with ones correlated to bandwidth usage. Also note that, in wireline networks, we assume each user to have a limited uplink budget (*e.g.*, K bytes per month). Hence, unfair resource

allocation can cause some users to consume their budget earlier, potentially leading to a noticeable decrease in the number of users that can participate in the cooperative delivery phase and thereby significantly limiting the future cooperation gains. For these reasons, our work puts greater emphasis on the fairness performance, which is defined based on Jain's fairness index [15].

In short, the proposed research in this paper applies the cooperation principle to IPTV networks to achieve the following objectives: (i) *maximize the servicing capacity in the network by minimizing the servicing overhead through the dedicated server*, and (ii) *increase the operational lifetime of the network by fairly distributing resources to session peers (i.e., clients that subscribe to the same multicast session)*, where the term *operational lifetime* refers to the timeframe on the clients' willingness or capability to cooperate. To achieve our objectives, we propose two cooperative content delivery techniques, Cooperative First-Come-First-Serve (C-FCFS) technique and Cooperative Weighted Fair Queueing (C-WFQ) technique.

We can summarize our contributions as follows:

- We present the design of a *peer-assisted server-based* on-demand content delivery framework for IPTV networks that explicitly supports fair resource allocation among session peers through adaptive resource management while introducing minimal servicing overhead at the server side. To maximize the peer participation ratio, the proposed framework limits cooperation to the *active peers* (*i.e.*, peers that are part of an ongoing session and actively participate the same session as both a receiver and a transmitter).
- We assign fairness-driven *utility weights* to each active peer for the duration of the content delivery phase to determine (i) the amount of content to be delivered by each peer and (ii) the transmission rate assigned to each of them. We dynamically adjust these weights to improve the fairness performance at the peers in regards to resource allocation by successively limiting the burden on the earlier arrived peers.
- We present an in-depth analysis of the proposed framework using three different request arrival models to measure the impact of correlated and/or bursty arrivals on the system performance. Through exhaustive simulation studies, we show that significant bandwidth savings can be achieved by effectively combining *asynchronous multicast* session deliveries with *patching unicast* services through session peers and the dedicated server.

The rest of the paper is organized as follows. In Section II we describe our system model. We discuss the algorithmic details of the proposed C-FCFS and C-WFQ techniques in Sections III and IV, respectively. We present the preliminary analysis of the proposed techniques in Section V. We investigate the impact of varying the request arrival rates on system performance in Section VI. We discuss the practical limitations for implementing the proposed techniques and our proposed modifications in Section VII. Section VIII introduces the related works, and we conclude our paper in Section IX.

II. System Model

We illustrate the basic architecture of an IPTV network in Figure 1. Within the given framework, Super Head-End (SHE) is the location where live content is acquired and real-time encoding of video broadcasts take place. The national content generated at the SHE is then forwarded over the core network to the regional Video Hub Offices (VHOs), which also provide real-time encoding services for the local broadcast and on-demand services. Content acquired/generated at the VHOs is then distributed to the local Video Switching Offices (VSOs), each of which forwards the incoming IPTV traffic to end-users over the access networks. In the current research, we focus on the operation at the VHO, which serves thousands of clients and which is responsible for the delivery of on-demand content.

Assume that there are n_j users that request session s_j's multicast within a certain timeframe, which we refer to as ΔT. The request times are given by $T_{req,ij}$, where $1 \leq i \leq n_j$ and $(T_{req,ij} - T_{req,1j}) \leq \Delta T$, $\forall i \in s_j$. Ideally, each user expects to join the targeted session's multicast at the time of its request. However, since the delivery of the on-demand content is typically initiated starting from the beginning, it becomes impractical to initiate a multicast session for each received request, especially when the sessions are long (*e.g.*, duration of 2 $hours$) and the requests arrive on a frequent basis.

To circumvent these limitations, we propose a cooperative delivery framework, for which the basic operations are shown in Figure 2. The proposed framework shown in Figure 2 utilizes a dedicated server (also referred to as the video server), which is essentially responsible for coordinating the received requests and allocating resources, and $n_j = k + 1$ clients each of which makes a join request for session s_j. Here, after the ith client makes a session-join request to the video server using the message Req_i, it also makes a request to join the source multicast for s_j. To guarantee an almost instant playout [1], we need to deliver the requested packets to each client at a rate that is at least equal to the delivery rate for the source multicast, which we refer to as W_M. In our framework, we also assume that the clients have a downlink capacity of $W_d > 2 \times W_M$, which allow the clients to support the simultaneous delivery of session multicast and unicast patch streams. Also note that, during the activity period of a session, resource availability at each session peer satisfies the following relationship based on the initially defined servicing requirements: a peer that arrives to the system at time t_i is assumed to have access to session data from the interval $(0, \delta)$ at time t_j, where $\delta \geq t_j - t_i$.

To deliver the requested packets in time, the proposed architecture utilizes the session peers first (using the *Pdata* stream) and the video server second (using the *Udata* stream for unicast transmissions). Note that, the server also uses the *Mdata* stream for its multicast transmissions (*i.e.*, session multicasts). Each client that has access to the requested content becomes a potential source for servicing the future

[1] We omit the brief delay (which equals almost one round trip time (RTT) between two peers) that is required to coordinate among session peers and deliver the requested resources from the session peers to a requesting client.

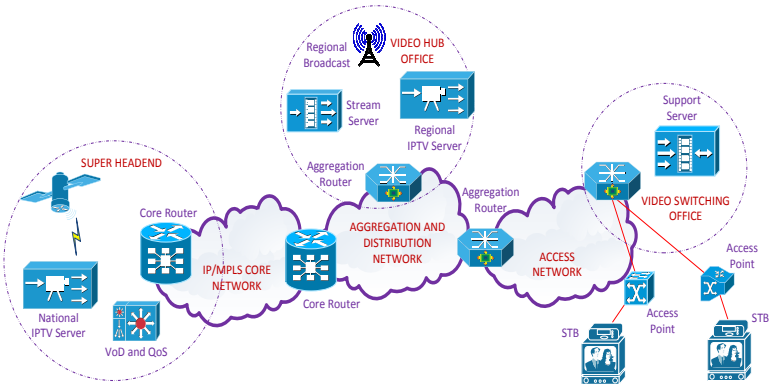


Fig. 1: Architecture of an IPTV network.

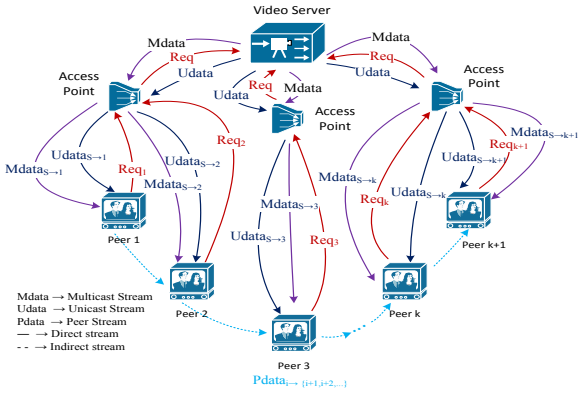


Fig. 2: Cooperative transmission framework.

requests. To communicate the delivery information to session peers, the dedicated server uses session-based multicast control channels and unicast control channels (which is also used for error recovery support). Multicast control channel is used for the delivery of commonly shared information (i.e., assigning unique identifier to peers and include the matching information in regards to IP addresses for the peers in the multicast control message). Unicast control messages are used to deliver information on the peer assignments.

In short, for all the received requests, there exists two main delivery sources, *i.e.*, session peers and the video server. However, as we mentioned earlier, video server is merely considered as an optional source, especially for the unicast-based deliveries. In our framework, video server is only used to compensate for the lack of resources at the client side, *i.e.*, when the session peers cannot provide the sufficient resources required to support the minimum delivery guarantees for the given request.

An important parameter of choice for the proposed architecture is the ΔT parameter, which represents the minimum waiting time for the content delivery server to initiate a new multicast session. As the duration of a multicast session increases, it becomes crucial to establish breathing instances in time, where the system can generate a new multicast session to accommodate for the needs of the newly arriving requests without violating the system constraints.

If the ΔT value is not properly assigned, clients may experience huge, and oftentimes unacceptable, session initialization delays. Choosing a small/large value for ΔT determines how efficiently the system resources are distributed among the session peers and the dedicated server. For instance, by choosing a larger value for ΔT, we would be increasing the burden at the client side, whereas, by choosing a smaller value for ΔT, we would be increasing the burden at the server side. Therefore, to optimize the resource usage efficiency in the network, we need to carefully study the relationship between request arrival process and the ΔT value, and evaluate the tradeoffs associated with its selection.

We next give an in-depth overview of the proposed cooperative content delivery techniques that aim to improve the resource usage efficiency in the network.

III. Cooperative First-Come-First-Serve (C-FCFS) Approach

C-FCFS approach is designed based on the following principle: *a session peer targets a single client and serves a single request at any given point in time and assigns a constant delivery period for each received request*. The reason why we refer to the proposed approach as C-FCFS is because the requests are serviced on a first-come-first-serve (FCFS) basis at each session peer.

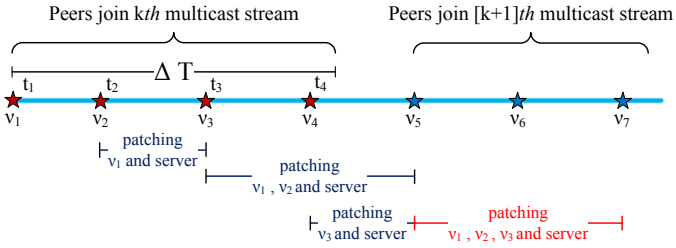


Fig. 3: Transmission events for the C-FCFS approach.

The basic operation for the C-FCFS approach is explained as follows. After the most recent join request, all session peers that have partial access to the requested content are asked to deliver an amount that is proportional to their initially assigned transmission weights (we will discuss this shortly). Since a session peer can only service a single request, any session peer that is busy responding to earlier received requests at the time the new request is received has to wait until it becomes idle. We illustrate the operation of the C-FCFS approach in Figure 3, which shows the join period for the kth and the $(k+1)$th multicast streams corresponding to the same session. As soon as the join request from ν_2 is received, patching service, which refers to the delivery of session data that the client has not yet received because of joining the multicast session late, is delivered through the dedicated server and the idle peer ν_1. When ν_4 makes a join request, patching service is initially delivered through ν_3 and the server, as ν_1 and ν_2 are servicing ν_3's request. However, as soon as ν_1 and ν_2 finish servicing ν_3's request, they can join ν_3 and the server to deliver the patching data to ν_4. At each instance, delivery rate through the server is adapted to match the servicing requirements, *i.e.*, if the delivery rate through the peers is greater than the minimum

required rate, then the server stops delivering patching service for the given request, and vice versa.

To enable fair access to the available resources, transmission weights are dynamically varied by taking into account the impact of previous transmissions, *i.e.*, the amount of data each user had delivered by the time the new request is received. Since any user, which makes a join request, needs to receive the synchronization data–the content delivered by the source multicast up until the time of request–with proper timing (*i.e.*, as if the transmissions are coming from a dedicated server that uses a unicast stream), time-varying delivery rates should allow the user to access the received session data at the original source multicast rate, W_M, that is, $D_{rcv}(t) \geq W_M \times (t - T_{req,j})$, where, $D_{rcv}(t)$ represents the amount of data delivered to the peer by t for the given request, $T_{req} \leq t \leq \big(2 \times T_{req} - T_1\big)$, and T_1 represents the start of transmission time for the multicast session targeted by the given request. The boundary conditions assume that, in the worst case, client receives the synchronization data at the source multicast rate.

Delivery rate requirements also suggest that, at any given point of time, the amount of data delivered to the user should be greater than or equal to the amount of data delivered by the hypothetical unicast stream through the dedicated server. Therefore, if the overall support through the session peers cannot satisfy the minimum delivery requirements, then we deliver the additionally requested resources through the video server.

In short, after client ν_i makes a join request to receive session s_j at time $T_{i,j}$, we calculate the resources that each session peer is expected to deliver as follows:

$$D_{k,i} = W_M \times (T_{i,j} - T_{1,j}) / \sum_{l=1}^{k-1} \big(w_l(T_{i,j}) / w_k(T_{i,j})\big) \quad (1)$$

where $D_{k,i}$ represents the amount of data that ν_k is expected to deliver in response to ν_i's request, $T_{1,j}$ represents the starting time for s_j's multicast, and $w_k(T)$ represents the weight assigned to ν_k at time T. Note that, in our framework, dedicated server initiates the multicast sessions reactively, *i.e.*, each multicast session starts with a new request.

To determine the user weights, we use the following equation:

$$w_k(T_{i,j}) = \left(\left\lceil 1 - B_k(T_{i,j}) / B_{j,\max} \right\rceil^+ \right)^{\alpha_{i,k}} \quad (2)$$

where $B_{j,\max}$ represents the maximum amount of data each session peer is allowed to deliver during the lifetime of s_j's multicast, $B_k(T_{i,j})$ represents the amount of data already delivered by ν_k by the time $T_{i,j}$, and $\alpha_{i,k}$ represents the normalization metric assigned to ν_k for ν_i's request. [2]

Note that, by varying the value of the parameter $\alpha_{i,k}$, we can determine the convergence speed for the user weights to the limiting value of zero. Using a higher $\alpha_{i,k}$ value allows a session peer reach the limit faster, thereby improving the fairness performance at the cost of increased server usage. Note that, we can dynamically update the normalization metric in response to changes observed in system state. In doing so, we can promptly react to sudden changes observed in, for instance, resource availability at the peers or the video server.

Equation (2) suggests that *session peers who have delivered the most by the time of request are given the least priority during the resource assignment process*. In doing so, we can gradually shift the main source of peer delivery away from the early arriving users and towards the late arriving users. As a result, we can improve the fairness among session peers. Additionally, if the peers observe varying uplink rates, the order of the packets delivered through the peers is updated by prioritizing the delivery of the earlier (or later) synchronization data segment through session peers with higher (or lower) uplink capacity.

IV. Cooperative Weighted Fair Queueing (C-WFQ) Approach

To further and more efficiently utilize the resources at the client side, for the second approach, we focus on the individual transmission subperiods (*i.e.*, each of which starts when a new request is received or the servicing of an earlier request finishes), instead of assigning a constant delivery period for each received request. Specifically, *each session peer targets all the active requests for the duration that they are allowed to service*, which depends on various factors, such as resource availability, resource usage history, etc. As the number of available session peers varies from one transmission subperiod to the next, so as the user weights and the expected contribution from each session peer.

At the beginning of each transmission subperiod, we update the user weights based on the earlier peer contributions for the given session. After each received request, we probe all the available peers to request their support. Since each session peer is allowed to service multiple requests at once, as in the case of Weighted Fair Queuing, we need to properly distribute the available resources to the active requests.

We illustrate the basic operation for the C-WFQ approach in Figure 4. Anytime a new request is received, each session peer reallocates its resources to accommodate the bandwidth requirements for the newly arrived request. In the given figure, when ν_1 and ν_2 receive a join request from ν_4, they immediately start to service ν_4's request while also continuing to service ν_3's request but at a lower servicing rate. Since ν_3 is only expected to service ν_4's request, it fully allocates its resources to service ν_4's request. We next explain in detail the operation of the proposed delivery framework.

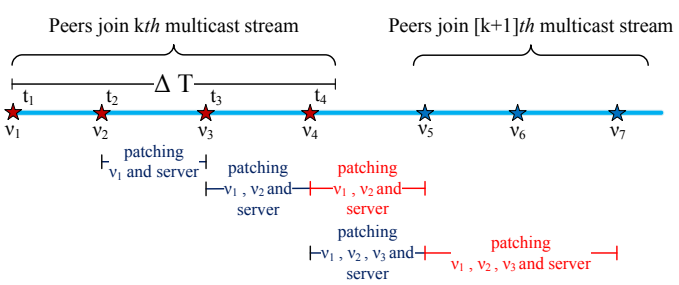


Fig. 4: Transmission events for the C-WFQ approach.

[2] $\lceil x \rceil^+ = \max(x, 0)$

Assume that user ν_i makes a request for session s_j at time $T_{i,j}$. At the beginning of each session, we reset the counter that represents the user count to 1, hence, by $T_{i,j}$, the number of requests that arrive for s_j–after it is created–is equal to $i-1$. For the time being, let us assume that users have unlimited transmission capacity along the downlink channel, allowing any number of session peers to be utilized during the synchronization phase. We can therefore use the following equation to find the weights for the peer contributions:

$$\omega_k(T_{i,j}) = \left(\left\lceil 1 - B_k(T_{i,j})/B_{j,\max} \right\rceil^+ \right)^{\beta_{i,k}} \tag{3}$$

where, for $k < i$, $B_k(T_{i,j})$ represents the contribution from ν_k by $T_{i,j}$ and $\beta_{i,k}$ represents the normalization metric for C-WFQ. Similar to the C-FCFS approach, we determine the size of expected delivery by each session peer using the following equation:

$$D_{k,i} = W_M \times (T_{i,j} - T_{1,j}) \sum_{l=1}^{i-1} \left(w_l(T_{i,j})/w_k(T_{i,j}) \right) \tag{4}$$

where $k < i$.

We next determine how to effectively distribute the available peer resources to multiple requests' use at each session peer. Since C-WFQ allows session peers to target multiple clients at once, if resource availability at a session peer is not sufficient to deliver the requested content at the desired rate, then a resource reallocation policy is needed to make the best use of available resources at the given peer. For that purpose, we use the following methodology.

We initiate the resource reallocation phase by finding the minimum required delivery rate for each session peer as follows:

$$W_{kl,\min}(T_{i,j}) = D_{k,l}(T_i, j)/(T_{l1,j} - T_{il,j}) \tag{5}$$

where $W_{kl,\min}(T_{i,j})$ represents the minimum required delivery rate for session peer ν_k in response to ν_l's request at time $T_{i,j}$, $D_{k,l}(T_{i,j})$ represents the updated value for the expected delivery from ν_k to ν_l at $T_{i,j}$, and $T_{li,j}$ equals $\left(T_{l,j} - T_{i,j}\right)$.

We next determine the total amount of resources that ν_k needs to allocate to each active request it has received:

$$W'_{k,\min}(T_{i,j}) = \sum_{l=k+1}^{i} W_{kl,\min}(T_{i,j}) \tag{6}$$

where $W'_{k,\min}(T_{i,j})$ represents the minimum required delivery rate by ν_k at $T_{i,j}$.

We can then calculate the initial rate assigned by ν_k to each active request it has received as follows:

$$W_{k,l}(T_{i,j}) = \begin{cases} \dfrac{\left(W_u^{(k)} - W'_{k,\min}(T_{i,j})\right)}{\sum_{\forall m} \left(\zeta^*_{k,m}/\zeta^*_{k,l}\right)} \\ +W_{kl,\min}(T_{i,j}) & \text{if } \dfrac{W'_{k,\min}(T_{i,j})}{W_u^{(k)}} < 1 \\ \dfrac{W_u^{(k)}}{\sum_{\forall m} \left(\zeta^*_{k,m}/\zeta^*_{k,l}\right)} & \text{otherwise} \end{cases} \tag{7}$$

where $W_u^{(k)}$ represents the available bandwidth along the uplink channel at user ν_k, and $\zeta^*_{k,l}$ is given by:

$$\zeta^*_{k,l} = \gamma_{k,l} \times W_{kl,\min}(T_{i,j}) \tag{8}$$

where $\gamma_{k,l}$ represents the weight assigned to ν_l's request at ν_k. Here, weight parameters are mainly used to prioritize certain requests, for instance, requests with shorter deadlines (*i.e.*, earlier received requests) or higher resource needs (*i.e.*, latest received requests). Due to space limitations, we assume $\gamma_{k,l} = 1$, suggesting that all the requests are assigned the same priority.

We next update the initially assigned weights based on the overall resource availability for each active request to further reduce the server's contribution. Specifically, assume that, for ν_l's request, the amount of available resources at the session peers is given by:

$$\Upsilon_l(T_{i,j}) = \sum_{\forall k \in C(l)} \widehat{W}_{kl,\min}(T_{i,j}) \tag{9}$$

where $C(l)$ represents the cooperation set for ν_l's request and $\widehat{W}_{kl,\min}(T_{i,j})$ is given by $W_u^{(k)} / \sum_{\forall m} \left(W_{ml,\min}(T_{i,j})/W_{kl,\min}(T_{i,j})\right)$

Assume that $W_{M,l}(T_{i,j})$ represents the minimum delivery rate requirement for ν_l's request at time $T_{i,j}$. When resources at the session peers are barely enough to service all the active requests, $W_{M,l}(T_{i,j})$ is expected to be equal to W_M. Since we aim to minimize the resource usage rate at the server side, we first determine whether $\Upsilon_l(T_{i,j})$ is less than or greater than $W_{M,l}(T_{i,j})$ and by how much its value differs from $W_{M,l}(T_{i,j})$ (*i.e.*, $\gamma_l(T_{i,j}) = |\Upsilon_l(T_{i,j}) - W_{M,l}(T_{i,j})|$). If, for instance, $\Upsilon_l(T_{i,j}) > W_{M,l}(T_{i,j})$, then we can allocate a portion of these resources, which amounts to $\gamma_l(T_{i,j})$ bits per second, to other requests' use without violating the service quality requirements for ν_l's request. Assume that Γ^+ represents the set of requests, for which the available resources at the session peers are more than sufficient to satisfy the service quality requirements (i.e., $\forall l \in \Gamma^+$, $\Upsilon_l(T_{i,j}) > W_{M,l}(T_{i,j})$), and Γ^- represents the set of requests, for which the available resources at the session peers are not sufficient to satisfy the service quality requirements (i.e., $\forall l \in \Gamma^-$, $\Upsilon_l(T_{i,j}) \leq W_{M,l}(T_{i,j})$). Then we can determine the total amount of resources that can be reallocated to other requests' use as follows:

$$\Upsilon^+ = \sum_{\forall l \in \Gamma^+} \left(\Upsilon_l(T_{i,j}) - W_{M,l}(T_{i,j})\right) \tag{10}$$

We can also determine the additional resources that would initially be requested from the server at $T_{i,j}$ as follows:

$$\Upsilon^- = \sum_{\forall l \in \Gamma^-} \left\lceil W_{M,l}(T_{i,j}) - \Upsilon_l(T_{i,j}) \right\rceil^+ \tag{11}$$

- If $\Upsilon^+ < \Upsilon^-$, then only a subset of the requests can be serviced without requiring additional assistance from the dedicated server.
- If $\Upsilon^+ \geq \Upsilon^-$, then resource requirements for all the available requests can be met without requesting additional

assistance from the dedicated server. [3]

In short, resources at the session peers are reallocated as follows:

$$W^*_{k,l}(T_{i,j}) = \begin{cases} \widehat{W}_{kl,\min}(T_{i,j}) \times \Bigg[1 - \dfrac{\min(\Upsilon^-, \Upsilon^+)}{\Upsilon^+} \\ \times \Big(1 - \dfrac{W_{M,l}(T_{i,j})}{\Upsilon_l(T_{i,j})}\Big)\Bigg] & \text{if } l \in \Gamma^+ \\ \widehat{W}_{kl,\min}(T_{i,j}) \times \dfrac{W_{M,l}(T_{i,j})}{\Upsilon_l(T_{i,j})} & \text{if } l \in \Gamma^- \end{cases} \tag{12}$$

After the resources are reallocated at the session peers, we need to also ensure that the local bandwidth constraints are met at each session peer. For instance, if at ν_k, $\sum_{\forall l} W^*_{k,l}(T_{i,j}) > W_{u,k}$, then we need to reduce the delivery rates assigned by ν_k to each of its active requests. In a similar way, if at ν_k, $\sum_{\forall l} W^*_{k,l}(T_{i,j}) < W_{u,k}$, then it becomes possible to assign more resources to the active requests at ν_k. For that purpose, we use the following procedure:

1. Set pseudo-active session peer set to U, where a *pseudo-active peer* refers to an active session peer that has additional resources to allocate for the currently active requests.
2. Update rates at each ν_k, where $\sum_{\forall l} W^*_{k,l}(T_{i,j}) \geq W_{u,k}$, by multiplying the assigned rates with $W_{u,k}/\sum_{\forall l} W^*_{k,l}(T_{i,j})$, and update set $U = U \setminus \nu_k$.
3. Check for final assignments at each active session peer and determine additionally needed resources by active requests. Specifically, $\forall l \in \Gamma^-$, find $\sum_{\forall k \in C(l)} W^*_{k,l}(T_{i,j})$ and check whether or not it is less than $W_{M,l}(T_{i,j})$. If it is, then keep request in Γ^-, otherwise $\Gamma^- = \Gamma^- \setminus l$.
4. Next, for all requests in Γ^-, find delivery ratio υ_l, which equals $\sum_{\forall k \in C(l)} W^*_{k,l}(T_{i,j})/W_{M,l}(T_{i,j})$.
5. Starting with the request in Γ^- that observes the lowest delivery ratio, $l = \min_k \upsilon_l$, reallocate resources for the given request at each session peer ν_m belonging to the set $C^*(l) = C(l) \cap U$, starting with peer that services the least number of requests in Γ^-. Resources are reallocated at each selected peer by borrowing resources from requests that experience a delivery rate greater than session multicast rate. Remove l from Γ^-.
6. Continue to reallocate resources by going back to Step 5 as long as the following is true: $C(\Gamma^-) \cap U \neq \varnothing$.
7. If additional resources are needed to meet service quality requirements for active requests, then allocate remainder of the resources through dedicated server.

Finally, resource allocation rate at each session peer is updated whenever the servicing of a request finishes and the resources used by the given request are released at the peer servicing the given request. For that purpose, a similar procedure is used to redistribute the released resources to the active requests starting with the request that receives resources through the server at the highest rate.

[3]Note that, the above statement is valid as long as the session peers have full access to the requested resources. For instance, if a peer has available bandwidth but not the content to service a given request, then we cannot transfer the peer's resources to service another request, thereby requiring the server to continue to assist the delivery process even though the overall peer bandwidth availability is greater than what is needed to service all the active requests.

TABLE I: Simulation Parameters

L_S	$100\ min.$
ΔT	$5 \times \kappa\ min.$, where $\kappa \leq L_S/5$
W_d	$25\ Mbps$
W_u	$1\ Mbps$
W_M	$3\ Mbps$
λ_S	$1/20\ reqs/sec$
$B_{\min}$	$2 \times L_S \times W_M\ bits$
$B_{\max}$	$2 \times \big(W_u/W_M\big) \times B_{\min}\ bits$
(α, β)	$(1, 1)$

V. Performance Analysis

In this section, we use a simulation-based study to investigate the performance of the proposed content delivery techniques when the request arrival process is modeled by using the (homogeneous) Poisson process. We list the simulation parameters in Table I, where, L_S represents the session length, ΔT represents the mean time spacing between two successive sessions, W_u (or W_d) represents the available servicing capacity along the users' uplink (or downlink) channel, W_M represents the source multicast rate, λ_S represents the request arrival rate, $B_{\min}$ represents the maximum amount of data that can be delivered-at the source multicast rate-during the activity lifetime of a session [4], and $\big(\alpha, \beta\big)$ represent the zero convergence rates for the assigned weights for the C-FCFS and C-WFQ approaches, respectively.

The value of $B_{\max}$ is chosen to ensure that client ν_i that transmits continuously during the lifetime of session s_j ends up with a worst-case scenario weight of $w_i(T_{1,j} + 2L_{S,j}) \geq 0$ by the end of s_j's activity lifetime, thereby limiting the peer contribution to $2 \times L_{S,j}/\rho_i$, where $\rho_i = W_M/W_{u,i}$. [5]

In our study, we mostly focus on the following performance measures: (i) average and maximum bandwidth usage at the server side, (ii) synchronization latency, which refers to the delivery time of unicast-transmitted session data (*i.e.*, patching stream), and (iii) fairness index for peer-bandwidth usage. Note that, to obtain the fairness results we use *Jain's fairness index*, which is defined based on the following equation [15]:

$$F_{\nu,j} = \frac{\Big(\sum_{i=1}^{N_j} B_i(\Delta T_j)\Big)^2}{N_j \times \sum_{i=1}^{N_j} B_i(\Delta T_j)^2} \tag{13}$$

where $F_{\nu,j}$ represents the level of fairness observed in bandwidth usage for peers connected to session s_j, and N_j represents the number of peers connected to session s_j during the same period (*i.e.*, Δ_T). The value of $F_{\nu,j}$ ranges from $1/N_j$ (worst-case scenario attained when only one peer delivers data) to 1 (best-case scenario attained when each peer delivers the same amount of data).

[4]Since a request that arrives at t needs to be serviced by $(2t - T_1)$, where T_1 represents the starting point for the session's multicast, the maximum delivery duration for a session is given by $2 \times L_S \times W_M$, when ΔT attains the highest possible value of L_S.

[5]The continual delivery period for a session peer is typically much smaller than $\big(2 \times L_{S,j}\big)$, since, as we will show shortly, the resource optimal value of ΔT is smaller than the value of $L_{S,j}$.

A. Results for the C-FCFS approach

We start our analysis by focusing on the impact of varying the ΔT parameter on the overall servicing overhead at the video server, for which the results are shown in Figure 5d, which depict the results for both the unicast-based and the multicast-based servicing overhead. Specifically, in Figure 5d we illustrate the results for the minimum required delivery rate at the dedicated server that meets the service quality requirements at the client side. As shown in Figure 5d, we achieve the resource optimal results when ΔT equals $\approx 14\ minutes$ (i.e., $\Delta T^* = 14$). In general, we can explain the relationship between servicing overhead and ΔT parameter as follows:

- $\Delta T < \Delta T^* \rightarrow$ the number of multicast streams for the given session is increased unnecessarily, *causing multicast portion of the server traffic to dominate the overall servicing overhead at the server side*.
- $\Delta T > \Delta T^* \rightarrow$ the desired usage rate for the session peers during synchronization phase is increased unnecessarily, *forcing the video server to increase its unicast-based servicing rate faster than it decreases the multicast-based servicing rate*.

Since the interarrival times between consecutive join requests are exponentially distributed, we can approximate the expected value for the unicast-based servicing overhead during a single activity period, which lasts for $2 \times \Delta T$, as follows [6]:

$$E[B_{S,u}] = W_M \mu_S (N^2 + N)(\rho - 1)^2 / (2\rho^2 + 2\rho) \qquad (14)$$

where μ_S represents the average time spacing between two successively received requests ($\mu_S = 1/\lambda_S$), ρ represents the average number of session peers required to service a given request and is given by the equation $W_M/E[W_u]$, and N is the average number of requests received during a single activity period (*i.e.*, $N = \Delta T/\mu$).

Using (14) we can find the average servicing rate for the received requests as follows:

$$E[W_S] \approx \frac{(1 + \Delta T/\mu_S)(\rho - 1)^2}{4(\rho^2 + \rho)/W_M} + \frac{W_M \times L_S}{2\Delta T} \qquad (15)$$

Figure 5a shows the results for $E[W_S]$, which represents the expected servicing overhead at the server side, as we vary the value of ΔT. The theoretical results shown in Figure 5a coincide nicely with our earlier simulation-based results shown in Figure 5d that assumed a value of 3 for the parameter ρ (*i.e.*, each peer has an uplink bandwidth capacity of $1\ Mbps$). We observe that the ΔT value that achieves the minimum servicing overhead decreases as we increase the value of ρ; and as the value of ρ goes to 1, optimal results are achieved when $\Delta T = L_S$. These results are expected since decreasing the value of ρ increases the effectiveness of delivering the content through the peers.

Next, we study the relationship between session length (L_S) and the resource optimal timing for the stream refresh instances, *i.e.*, $\Delta T^*/L_S$, for which the simulation-based results are shown in Figure 5e when $\rho = 3$. Using (14) we can approximate the value of ΔT^* (*i.e.*, the resource optimal value for ΔT) using $\min\left(\sqrt{2L_S\mu_S\rho(\rho+1)/(\rho-1)^2}, L_S\right)$.

We show the theoretical results for the ratio $\Delta T^*/L_S$ in Figure 5b as we vary the value of ρ. We observe that as we increase the value of L_S, the ratio of $\Delta T^*/L_S$ starts to converge, where the point of convergence is inversely proportional to the value of ρ. We also show the numerical values for the optimal ΔT parameter in Figure 5c. These results suggest that the value of ΔT that achieves the minimum servicing overhead at the server side introduces a fairly limited increase in overhead at the client side, since peer contributions decrease as we decrease the value of ΔT. Therefore, the selected ΔT^* value presents a good tradeoff point between server overhead and peer contributions. We observe that, compared to unicast-only delivery scenario, at the same ΔT^* value, the proposed approach achieves 68% improvement in the servicing overhead.

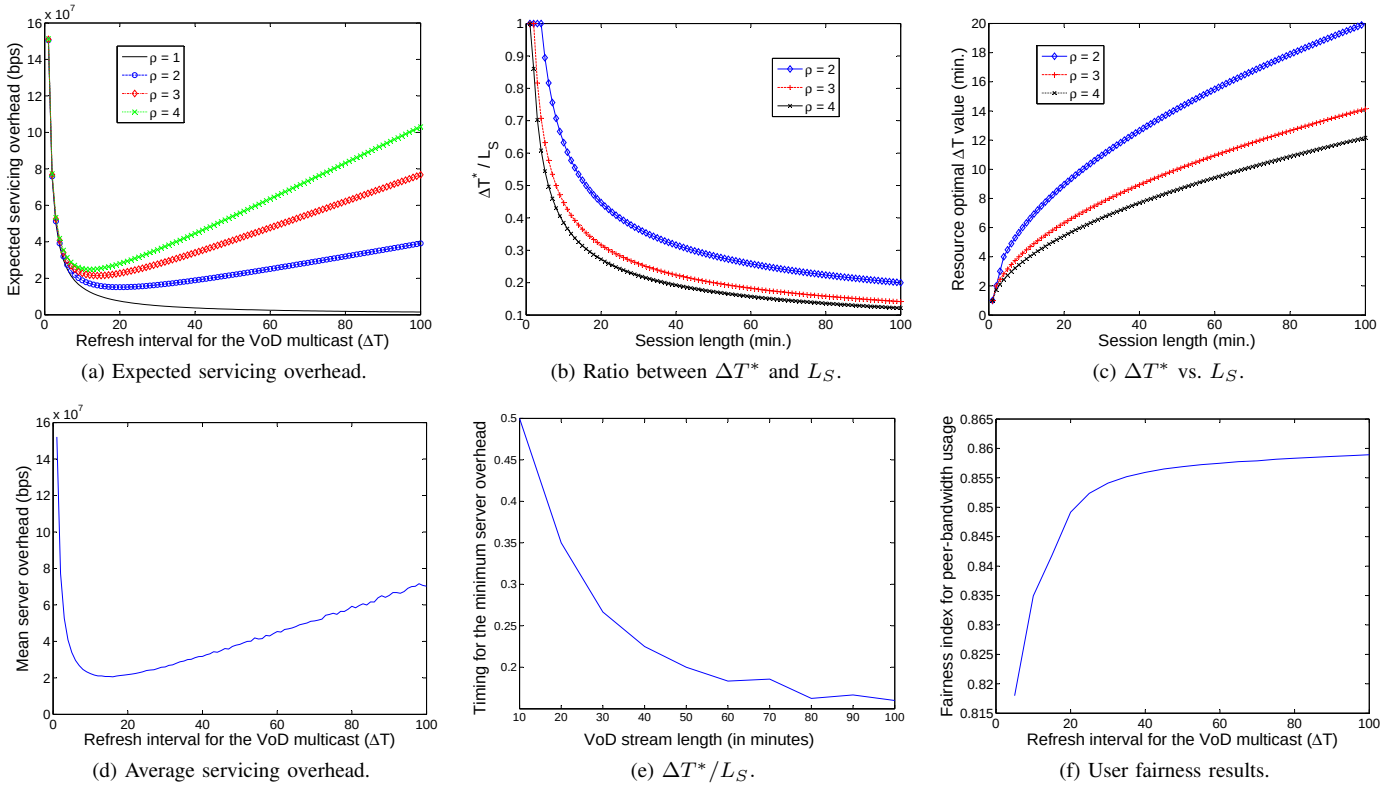
Another important statistical measure for the proposed framework is the distribution for the maximum bandwidth usage at the server side (per session), which illustrates the worst-case resource allocation scenario by the service provider. We need to carefully examine the given distribution so as to optimize the servicing cost at the server side, especially when multiple sessions are taken into consideration. We show the results in Figure 6a for the overall capacity usage (*i.e.*, the total bandwidth usage by the unicast and multicast streams). We observe that the results do not deviate significantly from the mean value, as we increase the value of ΔT, suggesting a stable performance.

We next study the relationship between maximum bandwidth usage at the server side and mean servicing overhead, for which the results are shown in Figure 6b. We observe that as we increase the value of ΔT, max-to-mean usage initially experiences a sharp increase in value, after which it starts to converge (to a value of 2.5, for the given scenario). The results also illustrate the relative impact of unicast-based and multicast-based overhead on the overall server capacity usage, *i.e.*, multicast traffic dominates at the smaller ΔT values, and unicast traffic dominates at the higher ΔT values. Also note that, since the max-to-mean server capacity ratio experiences a convergence in trend, depending on the chosen ΔT value, service provider can effectively use this relationship to pre-allocate its resources to different sessions with the objective of optimizing the overall capacity usage in the network and minimizing the overall servicing cost.

We next investigate the impact of ΔT parameter on the synchronization latency for the early processed requests, *i.e.*, requests that are serviced earlier than their corresponding deadlines, for which the results are shown in Figure 6c. In our simulations, we observed that (i) on average, approximately 18% of the received requests are serviced earlier than their respective deadlines, and (ii) for the earlier serviced requests, average delivery latency is reduced by 35%.

For the given framework, earlier servicing time typically suggests higher peer involvement for the received requests. Since dedicated server is essentially used to satisfy the minimum servicing requirements, earlier servicing time can only

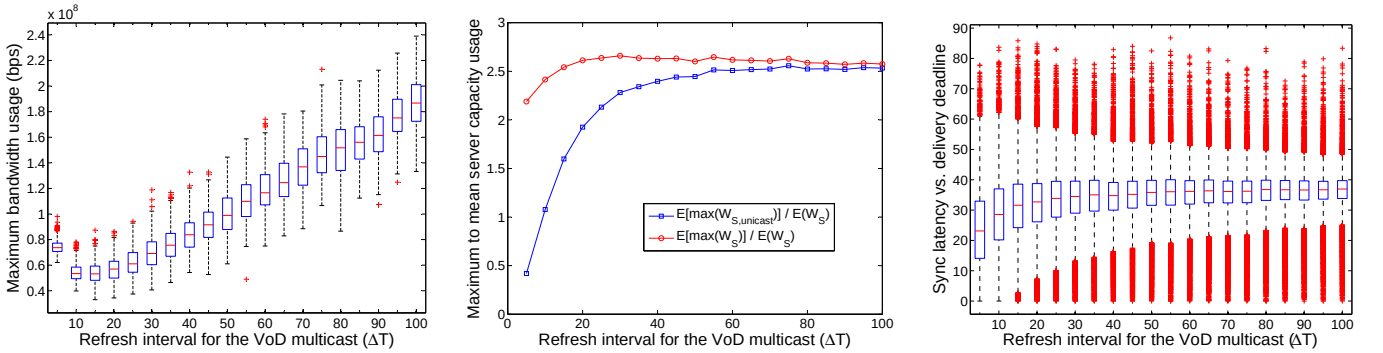
[6]For more detailed discussion on our analysis see Appendix.



(a) Expected servicing overhead. (b) Ratio between ΔT^* and L_S. (c) ΔT^* vs. L_S.

(d) Average servicing overhead. (e) $\Delta T^*/L_S$. (f) User fairness results.

Fig. 5: Expected and measured performances for C-FCFS.



(a) Distribution for the maximum servicing overhead. (b) Max-to-mean server capacity usage ratio. (c) Earliness measure for the processed requests.

Fig. 6: Overhead and latency performances for C-FCFS.

be achieved if we can satisfy $E[N_j] \geq W_M/W_u^{[j]}$, where N_j represents the number of session peers servicing the jth request and $W_u^{[j]}$ represents the average uplink bandwidth availability for the peers servicing the jth request.

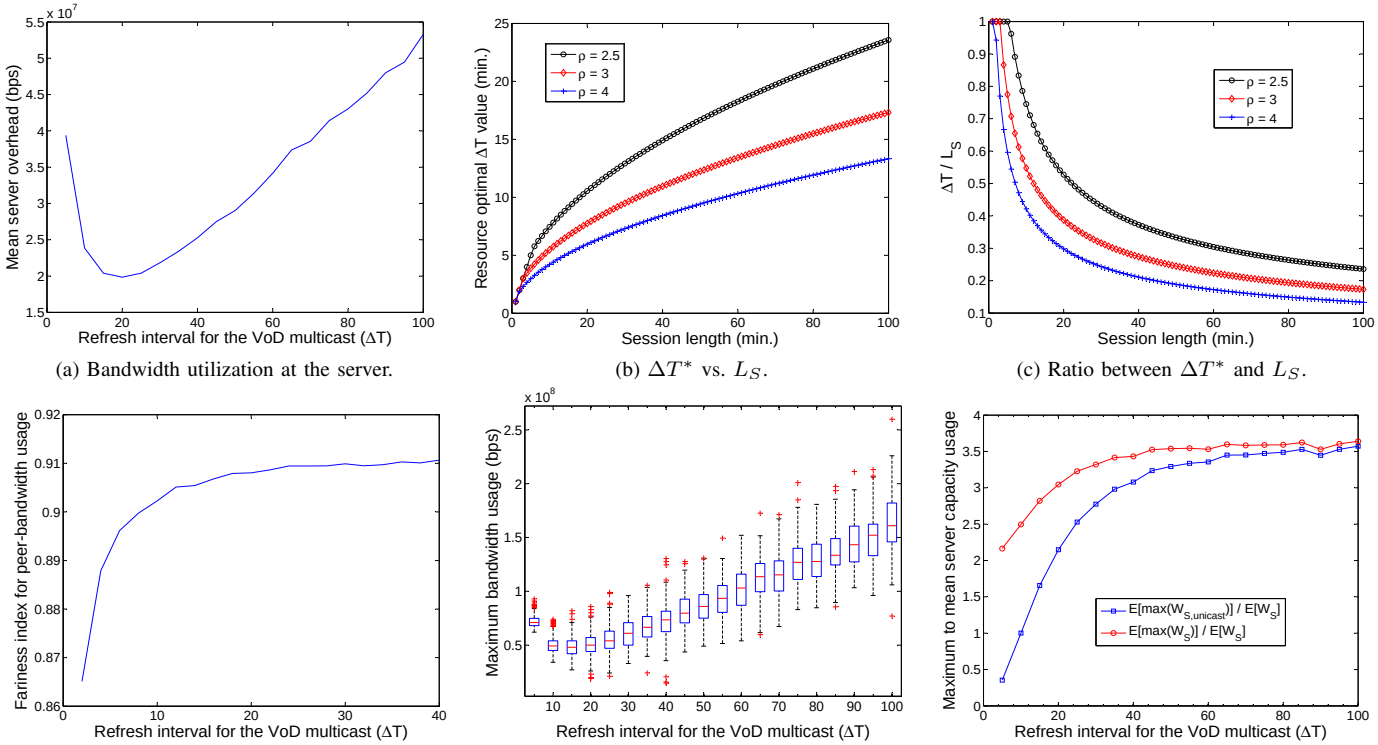
Since C-FCFS technique requires session peers to respond to the received requests one at a time and the requested content is delivered by session peers in a bursty delivery mode, as long as the overall resources at the session peers are more than sufficient to satisfy the minimum delivery rate guarantees, requested content can be delivered at a faster rate to the user making the request. As a result of this bursty delivery process, requests that are delivered earlier experience a noticeable decrease in the perceived delivery latency.

Lastly, we show the fairness performance for the C-FCFS technique in Figure 5f. In general, we observe acceptable results for the fairness performance, *i.e.*, $E[F_{\nu,j}] > 0.8$. The results improve as the value of ΔT_j increases (*i.e.*, the number of session peers that can actively contribute to the content delivery process increases). Consequently, user resources can be more evenly distributed to the incoming requests leading to the improvements observed in Figure 5f.

B. Results for the C-WFQ approach

In this section, we analyze the performance of the C-WFQ approach, when (homogeneous) Poisson process is used to model the request arrival process and the system parameters are chosen according to Table I.

We first investigate the overhead performance at the dedicated server. In Figure 7a we show the results corresponding to the average servicing overhead. Compared to the C-FCFS



(a) Bandwidth utilization at the server.

(b) ΔT^* vs. L_S.

(c) Ratio between ΔT^* and L_S.

(d) User fairness results.

(e) Distribution for the maximum servicing overhead.

(f) Max-to-mean server capacity usage ratio.

Fig. 7: Expected and measured performance statistics for C-WFQ.

approach, with the C-WFQ approach, we observe a noticeable decrease in resource utilization at the server side, especially at higher ΔT values. Since the C-WFQ approach utilizes session peers more efficiently, perceived improvement in the servicing overhead increases as the number of session peers increases (or as we increase the value of ΔT).

However, if we compare the resource optimal performances of the two approaches, we observe close results with minor differences. To be specific, when compared to C-FCFS, with C-WFQ, we observe $\approx 10\%$ improvement in the resource optimal servicing overhead. Furthermore, the optimal overhead is attained at a ΔT value that is $\approx 20\%$ higher than that of the C-FCFS approach. The reason for that is because, due to using peers more efficiently, the C-WFQ approach requires more peers to achieve the optimal overhead when compared to the C-FCFS approach. Consequently, we can utilize the resources at the peers more evenly, thereby leading to higher fairness index values as evidenced by the results shown in Figure 7d. Also note that, a higher ΔT^* value suggests higher flexibility in generating new session streams at the video server, thereby giving the server more freedom in the decisions it makes.

Using a procedure similar to the one we used for the C-FCFS technique, we can approximate the theoretical overhead for the C-WFQ approach using the following equation:

$$E[W_S] \approx \frac{W_M}{4\rho^2} \times \left[\frac{(\rho-2)(\rho-1)}{\mu/\Delta T} + \left(2\rho^2+\rho-4\right) + \frac{\rho^2(L_S-2\mu)+\mu(3\rho-2)}{\Delta T/2} \right] \quad (16)$$

which is valid when $\rho > 2$. [7]

Then, using (16) we can approximate the value for ΔT^* using $\Delta T^* \approx \min\left(\sqrt{\frac{2L_S\rho^2/\lambda_S}{(\rho-2)(\rho-1)}}, L_S\right)$

In Figures 7b and 7c, we illustrate the relationship between the resource optimal ΔT^* parameter and the session length. When compared to the theoretical results for the C-FCFS approach, we observe a noticeable increase in the value of ΔT^* ($\approx 20\%$ increase).

Since the C-WFQ approach allows session peers to distribute their resources more efficiently (when compared to the C-FCFS approach), minimum required servicing rate at the server side increases at a slower rate. If we compare the resource optimal ΔT^* values for both approaches, we observe that the function $r_\Delta(\rho)$ which is defined as $\Delta T^*_{c-wfq}(\rho)/\Delta T^*_{c-fcfs}(\rho)$ and which approximately equals $\sqrt{\frac{\rho\times(\rho-1)}{(\rho-2)\times(\rho+1)}}$ shows the characteristics of a long-tailed distribution. To be specific, the ratio $r_\Delta(\rho)$ initially experiences

[7] We require different approximations at lower ρ values. For the sake of simplicity, and since we are mostly interested in low peer uplink bandwidth availability, we only present the results that correspond to the scenarios with $\rho > 2$.

a sharp decrease as we increase the value of ρ (*i.e.*, decrease from 2 at $\rho = 2.2$ to 1.225 at $\rho = 3$), after which it shows a slower decline in value eventually converging to 1 (when both of their values converge to $0.1 \times L_S$, which equals 10 $minutes$).

Next in Figure 7e we show the variations observed in the maximum bandwidth usage at the server side. Similar to the above results, we observe noticeable improvement in the servicing capacity usage due to more efficient use of session peers. However, unlike the C-FCFS approach, the relative ratio between the maximum capacity usage and mean servicing overhead is much higher for the C-WFQ approach, as shown in Figure 7f. That is because the worst-case performances of the two approaches converge as we increase the value of ΔT.

We also observe that (when we compare equations (15) and (16)) as we increase the value of ρ, the overall advantage of C-WFQ over C-FCFS in regards to the optimal overhead decreases, with both approaches attaining the same value when ρ is within the range of $(5.7, 5.9)$. As we increase the value of ρ further, we observe the C-FCFS approach to slightly outperform the C-WFQ approach ($\approx \%2$). These results are expected, since the overall bandwidth availability decreases as we increase the value of ρ, and we observe less advantage in distributing the highly limited peer resources to many requests' use.

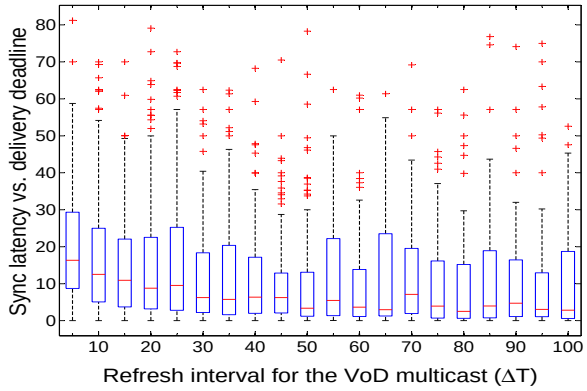


Fig. 8: Earliness measure for the processed requests ($L_S = 100m$).

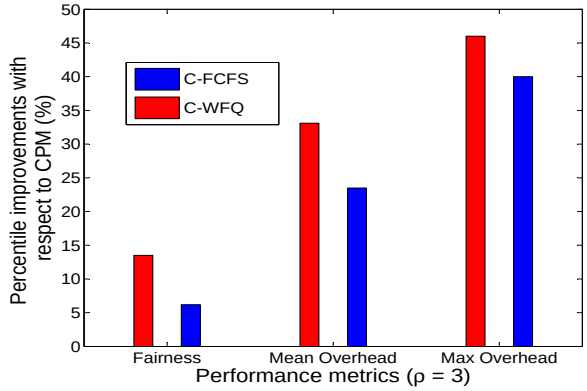


Fig. 10: Performance improvements when compared to CPM.

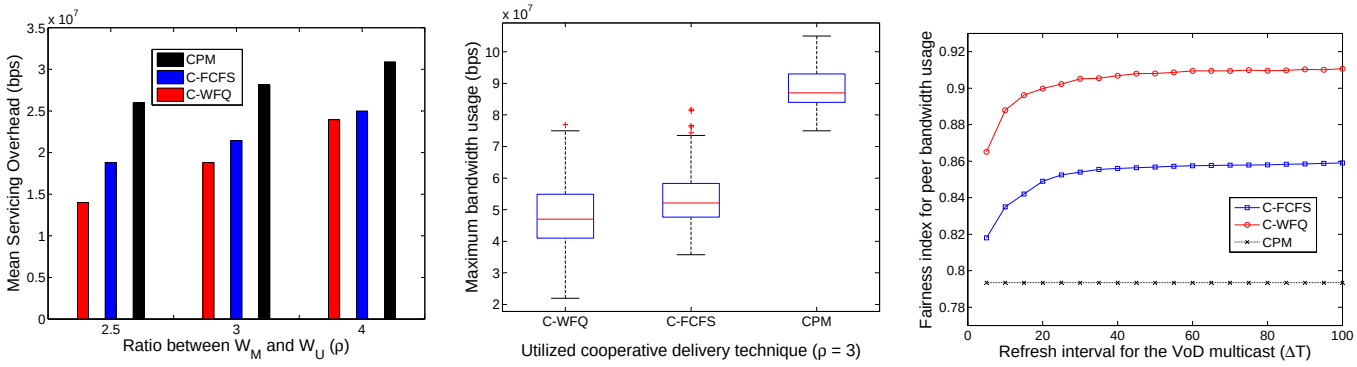
Lastly, we investigate the synchronization latency performance of the C-WFQ approach. In our simulations, we observe that less than 1% of requests are delivered earlier to the clients. Specifically, we find that on average $9.26 \times 10^{-2}\%$ of the received requests are delivered earlier, when the C-WFQ approach is used. In other words, almost all the requests are delivered at the request deadline, which coincides with the objectives stated for the C-WFQ approach. Specifically, since the C-WFQ approach aims to deliver the requested packets within the deadline by distributing peer resources to multiple requests at once, oftentimes minimal resources are utilized (at the peers) for each active request to allow for efficient servicing of the received requests.

Additionally, for requests that are processed earlier, Figure 8 shows the variations observed in the perceived latency values as we vary ΔT. We observe that as we increase the value of ΔT, we reduce the synchronization latency (from $\approx 15\%$ at $\Delta T = 5$ $min.$ to $\approx 3\%$ at $\Delta T = L_S$), which is expected as increasing the value of ΔT also increases the total amount of data that needs to be delivered per session peer. As the session peers hit the the uplink capacity barrier faster at higher ΔT values, session peers start to reduce the bandwidth they allocate per request to a value around or less than the minimum required servicing bandwidth for the given request. Consequently, with the help of the dedicated server, we can service most of the received requests at the minimum required delivery rate.

C. Performance comparisons to Cooperative Peer-assist and Multicast (CPM)

In this section, we compare the performance of the proposed cooperative delivery techniques to that of Cooperative Peer-assist and Multicast (CPM) approach proposed in [16], which divides each session into multiple chunks of 30-second duration. We briefly explain the operation of CPM as follows (as it is implemented in our simulation framework): (i) each session starts with a unicast request to the content delivery server (CDS) to receive the first chunk immediately from the server, (ii) clients continue to make requests for the following chunks as long as they have sufficient bandwidth availability along the downlink channel, (iii) if a viable multicast session is already scheduled for the requested chunk, then client is added to that multicast session, (iv) otherwise, if there are peers with direct access to the requested chunks (through caching), server responds with information on those peers (*i.e.*, a small randomly chosen set of peers) and the client makes its request to the received set of peers using a randomized order, (v) and, if no available peer is found to respond to the request, a new multicast session is scheduled based on the delivery deadline for the received request.

Additionally, we made the following assumptions for CPM to make its operation compatible with our framework: (i) no pre-caching is allowed, *i.e.*, users are only allowed to request chunks from CDS or peers connected to the same session, (ii) the number of pending requests is limited by the bandwidth availability at the client side. Due to space limitations, we will present the results on the most crucial performance measures, *i.e.*, mean server overhead, fairness measure, and the maximum bandwidth requirements at the server side.



(a) Bandwidth utilization (b) Maximum servicing overhead (c) User fairness

Fig. 9: Performance comparison between CPM and the proposed techniques.

In Figure 9a we show the results for the mean servicing overhead, when each proposed technique uses the resource optimal refresh period (ΔT^*). We observed an improvement of 24% with the C-FCFS technique and 33% with the C-WFQ technique, when ρ equals 3. We also observed that the improvements increased (or decreased) as the value of ρ is decreased (is increased). These results suggest that our techniques utilize the peers more efficiently than CPM. We continued to observe an improvement in the servicing overhead even as we reduce the bandwidth availability at the client side. Note that, the reason for the decrease in the improvement can be explained with the fact that the overall contributions through the peers become less comparable to the actual bandwidth requirements to service the clients' requests in a timely manner.

Next in Figure 9b we present the results for the maximum bandwidth usage when ρ equals 3. We observed on average 40% (in the case of C-FCFS) to 46% (in the case of C-WFQ) improvements for the maximum bandwidth requirement at the server side when compared to CPM. Since server side resources are typically provisioned based on the worst-case scenario for user demands, our results suggest significant reduction in the service provisioning costs. The increased performance improvements can be explained with the way the resources are requested from the server. In CPM, resources for future chunks are requested earlier to meet the deadline requirements, allowing multicast sessions to be established earlier to service future requests, limiting the use of peers, especially when the chunk duration is long.

The next set of results, which are illustrated in Figure 9c, focus on the fairness performance (when ρ equals 3). We observe an improvement of 6.2% (in the case of C-FCFS) to 13.5% in fairness performance when compared to CPM. Since our techniques adjust the level of peer contributions on a regular basis during the run of a session, requested resources can be distributed more evenly to the session peers, allowing high usage peers to reduce their contributions as more peers join the system.

Figure 10 summarizes our results and shows the percentile-based improvements for each approach separately with respect to CPM, validating our claims in regards to the resource usage efficiency for the proposed cooperative delivery techniques.

VI. Impact of Varying the Request Arrival Process

In Section V, we analyzed the performance of the proposed techniques using the Homogeneous Poisson Process (HPP) to generate independent and identically distributed arrival events for session join requests. However, in IPTV networks, requests typically arrive to the system based on a time-varying and bursty arrival process (*i.e.*, different arrival rates at different times of the day and batch arrivals at the start of the hour and/or half-hour) [12].

In this section, we take into account these characteristics of IPTV networks to investigate the impact of utilizing more generalized request arrival processes on the performance of the proposed techniques, and compare the results to that of HPP-based request arrival process.

In the following study, we specifically focus on two arrival processes: *Inhomogeneous (or Non-homogeneous) Poisson Process* (IPP) to illustrate the impact of time-varying arrival rates, and *Markov Modulated Poisson Process* (MMPP) to evaluate the impact of traffic burstiness.

A. Inhomogeneous Poisson Process

To generate the parameters for the Inhomogoneous Poisson Process we use the following methodology. We first use a two-state Markov model to determine the state transitions for the request arrival rates, *i.e.*, whether to increase it or to decrease it. For that purpose, we define two states, referred to as S^+_{IPP} and S^-_{IPP}, to represent these transitions, *i.e.*, if the system is in state S^+_{IPP} we increase the arrival rate, and if the system is in state S^-_{IPP} we decrease the arrival rate.

In our model, we also assume that the selected rate stays constant for a period of T_{IPP} (where T_{IPP} is assumed to be equal to one second in our analysis). We then initialize the request arrival rate to 1 and set the initial state to S^+_{IPP}. We next implement the following steps recursively for the duration of our simulation:

- When the system is in state S^+_{IPP}, we increase the request arrival rate by λ^+_{IPP} (*i.e.*, $\lambda(t) = \lambda(t-1) + \lambda^+_{IPP}$).
- When the system is in state S^-_{IPP}, we decrease the request arrival rate by λ^-_{IPP} (*i.e.*, $\lambda(t) = \lambda(t-1) - \lambda^-_{IPP}$).
- To also ensure that the selected rate stays within certain boundaries, (*i.e.*, $\lambda_{IPP,\min} < \lambda(t) \leq \lambda_{IPP,\max}$) we apply

the following check after each update: if the current state is S^{+}_{IPP}, then $\lambda(t) = \min(\lambda(t), \lambda_{IPP,\max})$; and if the current state is S^{-}_{IPP}, then $\lambda(t) = \max(\lambda(t), \lambda_{IPP,\min})$.

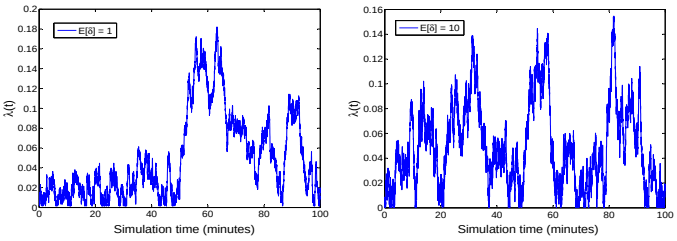
For $\lambda_{IPP,\min}$ we assumed a value of 1 and for $\lambda_{IPP,\max}$ we assumed a value that equals the desired ratio between the maximum and the minimum rates (*e.g*, 15 or 20). [8]

After the preliminary values for $\lambda(t)$ are determined to represent the arrival rates for the duration of the simulation run, we can finalize the request arrival rates as follows:

$$\lambda_{IPP}(t) = \overline{\lambda}_{IPP} \times \lambda(t) / \sum_{\forall \tau} \lambda(\tau) \qquad (17)$$

where $\overline{\lambda}_{IPP}$ represents the mean request arrival rate to the dedicated server.

1) Initialization Phase: To initialize the values for the time-varying arrival rates, we use the following arbitrarily chosen parameters for the 2-state Markov model: $p = 0.5$ and $q = 0.5$, where p represents the probability of switching from state S^{+}_{IPP} to state S^{-}_{IPP}, and vice versa. Furthermore, we varied the values for the λ^{+}_{IPP} and λ^{-}_{IPP} parameters between 1 and 10. Assuming that the values for λ^{+}_{IPP} and λ^{-}_{IPP} are drawn from the same distribution, we can represent their values using a single measure, which we refer to as δ. Note that, the selected input parameters create frequent transitions in the perceived arrival rates, for which the rate of change is proportional to the selected value for the δ measure, as shown in Figure 11a when $E[\delta] = 1$, and in Figure 11b when $E[\delta] = 10$.



(a) Slowly-varying case (b) Fast-varying case

Fig. 11: Arrival rates for different IPP processes.

2) Results for the C-FCFS technique: We start by analyzing the distributions for the maximum servicing overhead at the dedicated server, for which the results are shown in Figure 12. Compared to our earlier results, we observe up to 60% increase in the maximum overhead at the dedicated server with IPP-based arrivals. We observe similar results for the average servicing overhead as shown in Figure 13, with $\approx$ 20% increase in the resource optimal bandwidth requirements.

These results are expected as the time-varying arrival rates create occasional bursts, during which the server is probed more frequently. Furthermore, we observe in Figure 14 that the ratio between the maximum and the mean overheads increases from ≈ 2.5 to ≈ 2.7 suggesting that the use of time-varying arrival rates causes the requirements on the maximum servicing capacity to increase faster than the requirements on the mean servicing bandwidth.

[8]If it is also required to change the rates at each state transition point, then we can implement an additional update on the system state when the $\lambda(t)$ becomes equal to the boundary points, *i.e.*, when $\lambda(t) = \lambda_{\min}$ change the system state at time t to S^{+}_{IPP}, and when $\lambda(t) = \lambda_{\max}$ change the system state at time t to S^{-}_{IPP}.

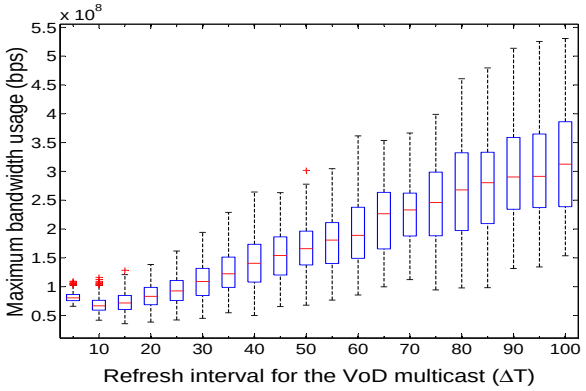


Fig. 12: Maximum joint unicast/multicast servicing overhead for C-FCFS.

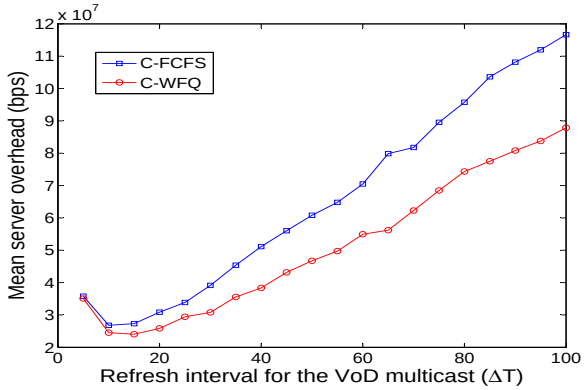


Fig. 13: Average servicing overhead with IPP-based arrivals.

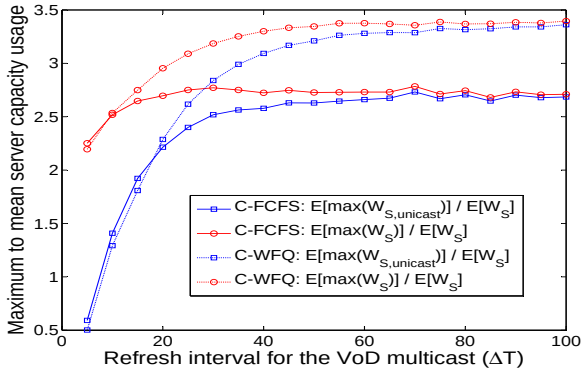
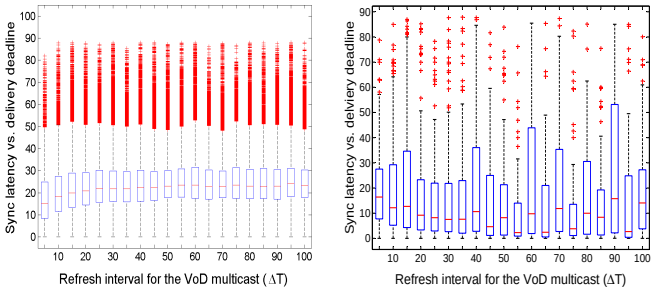


Fig. 14: Max to mean ratio for the servicing overhead with IPP-based arrivals.

Next we analyze the latency performance. The results are shown in Figure 15a. We observe that 17.6% of the requests are serviced earlier. As the number of requests, which require explicit support from the server side, increases, we observe an increase in the number of requests that are serviced at the minimum required rate, instead of the minimum available rate (which is typically higher than W_M, if the clients are the main source of delivery). As a result, the latency performance also degrades, as the requests are serviced later. For instance,

compared to the HPP-based scenario, when $\Delta T = L_S$, the earliness measure decreases from $\approx 35\%$ to $\approx 24\%$.



(a) C-FCFS (b) C-WFQ

Fig. 15: Earliness measure for the processed requests.

Lastly, we investigate the fairness performance of the C-FCFS approach for the given scenario. The results are shown in Figure 16. The results are very similar to that of the HPP-based scenario, suggesting that the time-varying arrival rates have negligible impact on the way the resources are allocated at the session peers.

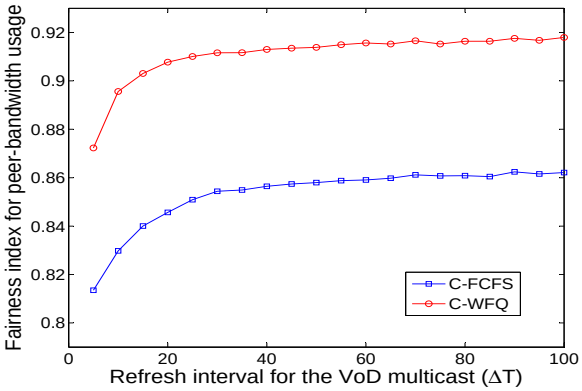


Fig. 16: User fairness with IPP-based arrivals.

3) Results for the C-WFQ technique: We next present the results for the C-WFQ approach. In Figure 17 we show the results for the maximum servicing overhead at the server side, and Figure 13 shows the results for the average servicing overhead.

The overhead performance is consistent with that of the C-FCFS approach. We observe a significant increase in the maximum servicing overhead due to the use of IPP-based request arrivals. The main difference between the two approaches is that the mean servicing overhead increases at a faster rate when compared to the increase observed in the maximum servicing overhead. As shown in Figure 14 maximum to mean server capacity usage ratio drops to 3.4 (for the IPP-based scenario) from 3.6 (for the HPP-based scenario).

Since time-varying bursty arrivals limit the contribution from the session peers, regardless of the approach we use, during the periods where the requested resources significantly exceed the level of support offered by the session peers, the amount of resources requested from the server side stay around the same level. That is why, we observe around 35 $Mbps$ increase in average bandwidth requirements at the server side for both of the given approaches. Since the C-WFQ approach performs better in more typical scenarios, such level of increase has a more significant impact on the overall results.

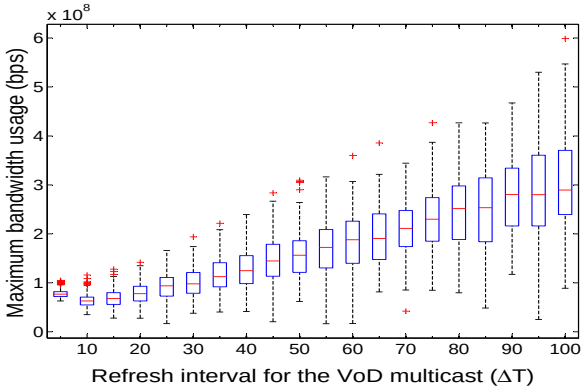


Fig. 17: Maximum servicing overhead for C-WFQ.

Lastly, we observe similar results for the latency and fairness performances. We find that the number of requests that are delivered earlier continue to stay small (around $8.87 \times 10^{-2}\%$), except that we observe higher variance in the earliness measure values, as shown in Figure 15b. Furthermore, as shown in Figure 16, we continue to observe very good fairness results as the fairness measure continues to stay around the value 0.91.

B. Markov Modulated Poisson Process

We next study the impact of using a Markov Modulated Poisson Process (MMPP) based arrival process on the performance of the proposed content delivery techniques. To generate the MMPP-based arrivals, we use a discrete-time two-state ON-OFF model where the arrivals occur during the ON state based on the underlying Poisson process, and no arrivals occur during the OFF state. We illustrate the state transitions in Figure 18, where the parameters p and q represent the probability of making a transition from ON state to OFF state and from OFF state to ON state, respectively. Here, by varying the values for the $\{p, q\}$ parameters, we can adjust the level of burstiness observed in the incoming traffic.

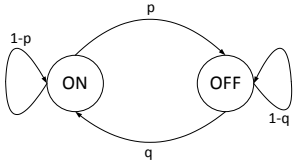


Fig. 18: State transitions for the two-state on-off model.

We can summarize the process we use to generate the arrival events as follows.

- First, we divide the simulation period into 1 $second$-long observation periods.
- Next, we generate the state transition events using the selected values for the $\{p, q\}$ parameters.
- Lastly, we use the information on the underlying Poisson process for the ON state to generate the arrival events for the join requests.

In our simulations, to study the basic performance measures, we used two different set of values for the $\{p, q\}$ parameters that correspond to mid-level and high-level burstiness in the incoming traffic.

The first model, which we refer to as $mmpp_1$, uses the following values: $\{p, q\} = \{0.06, 0.01\}$, which suggests, on average, 7 times increase in the request arrival rate during the ON state. The second model, which we refer to as $mmpp_2$, uses the following values: $\{p, q\} = \{0.05, 0.005\}$, which suggests an 11 times increase in the request arrival rate during the ON state.

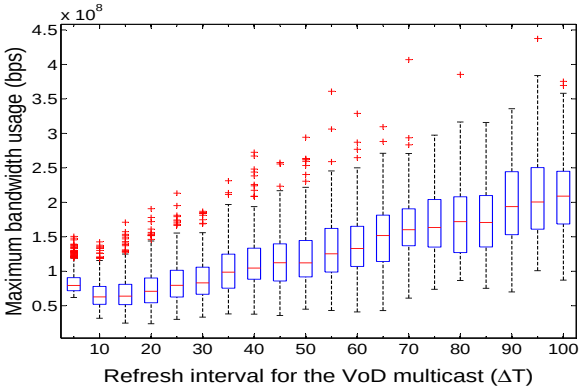


Fig. 19: Maximum servicing overhead for C-FCFS with $mmpp_1$-based arrivals.

1) Results for the C-FCFS technique: We first analyze the overhead performance under the $mmpp_1$ scenario. We show the results for the maximum servicing overhead at the server side in Figure 19. We observe that the system reacts to the arrival of bursty traffic better, when the arrival events are limited to occur at specific periods. We still observe an increase in overhead compared to the HPP-based scenario, however, the performance degradations are limited to 25% instead of 60% that we observed for the IPP-based scenario.

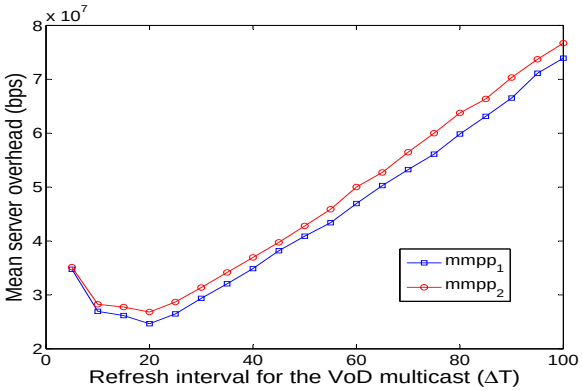


Fig. 20: Average servicing overhead for C-FCFS with $mmpp$-based arrivals.

Furthermore, we observe very little change in the average overhead performance, for which the results are shown in Figure 20. For instance, we observe $\approx 10\%$ increase in the minimum required overhead, which is achieved when $\Delta T^* = 0.2 \times L_S$. To validate these results, in Figure 21 we also investigate the impact of ΔT on the maximum to mean server capacity usage ratio. We observe that the max-to-mean ratio converges to 2.8 for the given scenario, suggesting a more controlled increase in the average overhead requirements.

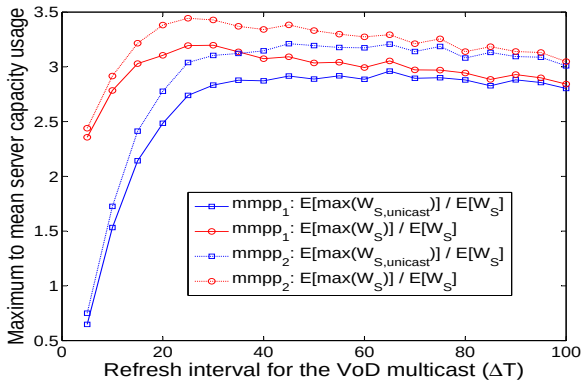


Fig. 21: Max-to-mean ratio for servicing overhead for C-FCFS with $mmpp$-based arrivals.

We next analyze the performance of the C-FCFS technique under the $mmpp_2$ scenario. In Figure 22, we show the results for the maximum server capacity usage. As expected, the overhead performance degrades as we increase the burstiness ratio. We observe that compared to the $mmpp_1$ scenario, the average value for the maximum required overhead increases by $\approx 10\%$. This is essentially caused by the noticeable increase we observe in the range of values corresponding to the maximum capacity usage (around 50%).

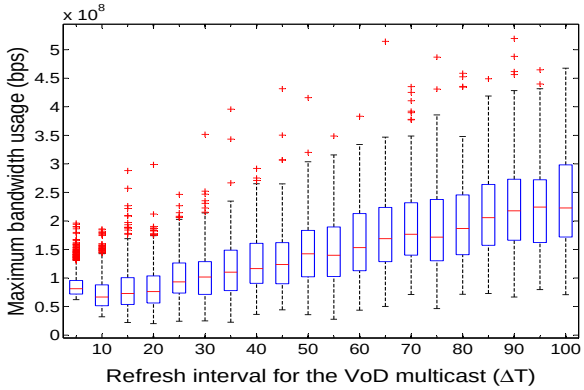


Fig. 22: Maximum servicing overhead for C-FCFS with $mmpp_2$-based arrivals.

We also observe a similar trend in the average overhead performance, for which the results are shown in Figure 20. However the increase is less noticeable when compared to the maximum overhead performance (less than 10% increase at the resource optimal ΔT value, and less than 5% when $\Delta T = L_S$).

We show the results corresponding to the the max-to-mean ratio performance for the servicing overhead in Figure 21, which suggests a limiting value of 3, when $\Delta T = L_S$. Since the mean arrival rate is kept constant, increasing the burstiness also increased the duration of the OFF periods. As a result, session peers had more time to service their requests and, in doing so, prevent the server from over allocating its resources.

We lastly want to comment on the latency and fairness performances. The results are very close for both scenarios and suggest negligible changes for the given performance measures (*i.e.*, for both arrival scenarios, approximately 18.5% of the requests are serviced earlier, with an average earliness measure of 35%). The main difference between these two scenarios is that increasing the burstiness level caused a decreasing trend in the fairness performance. Since the requests oftentimes arrive in batches, session peers have less opportunity in time to allocate their resources in a more distributed manner. As a result, session peers are not utilized at the desired rates, which in turn lowers the perceived fairness index values.

2) Results for the C-WFQ technique: We show the results for the mean servicing overhead under $mmpp_1$ arrival scenario in Figure 23, which suggests a noticeable improvement when compared to the HPP-based arrival scenario: we observe 10% improvement in the resource optimal overhead, and $\approx 15\%$ improvement in overhead when $\Delta T = L_S$.

Furthermore, as shown in Figure 24, the worst-case performance for the servicing overhead also improves (by more than 20%) when compared to the HPP-based arrival scenario. As a direct consequence of this, max-to-mean server capacity usage ratio reduces to ≈ 3.4 (from ≈ 3.6) as evidenced by the results shown in Figure 25.

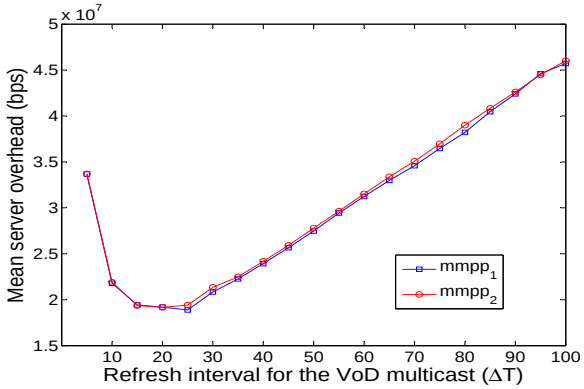


Fig. 23: Average servicing overhead for C-WFQ with $mmpp$-based arrivals.

We can use a similar reasoning to that of the C-FCFS approach to explain the cause of the perceived performance improvements. As the session join requests arrive in shorter timeframes separated by longer idle periods, session peers can serve the received requests more efficiently. Because of the longer breaks in between bursts, peers can allocate more resources per request. As a result, server is utilized at a lower rate on average.

When we increase the burstiness level, we observe similar results for the average servicing overhead, for which the results are shown in Figure 23. The main difference between the two scenarios is that the resource optimal overhead is attained at a lower ΔT value.

On the other hand, we observe a noticeable increase in the maximum servicing capacity usage, specifically at higher ΔT values (*i.e.*, when $\Delta T > 25$), as the perceived performance is affected the most by the increase in the burstiness level.

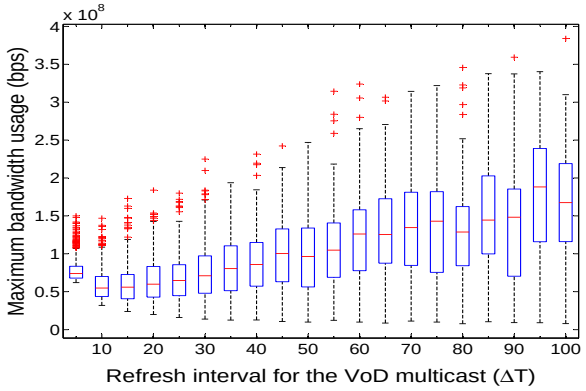


Fig. 24: Maximum servicing overhead for C-WFQ with $mmpp_1$-based arrivals.

As shown in Figure 26, compared to the $mmpp_1$ scenario, we observe $\approx 25\%$ increase in maximum capacity usage at the server side, which also causes a significant increase in the converging value for the max-to-mean usage ratio, from ≈ 3.4 to ≈ 4, as shown in Figure 25.

These results are expected as the increased number of requests received during the ON periods also increase the burden on the server side as it becomes more difficult for the session peers to service them at the desired rate. However, on average such increase in maximum server usage capacity is compensated by the increased length of the OFF periods, allowing peers to finish up their requests faster.

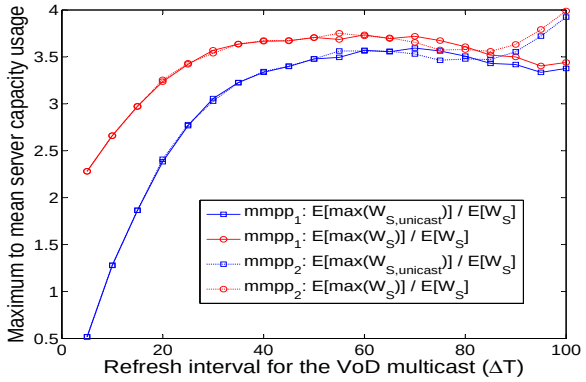


Fig. 25: Max-to-mean ratio for servicing overhead for C-WFQ with $mmpp$-based arrivals.

Consequently, for both scenarios, we observe a noticeable increase in the number of requests that are serviced earlier. Specifically, we observe ten times increase in the number of requests serviced earlier with the $mmpp_1$ and $mmpp_2$ scenarios (around 0.8%) when compared to the HPP- and IPP-based arrival scenarios. Furthermore, we observe that the earliness index for these requests also increases to $\approx 20\%$.

Lastly, for all the considered ΔT values, we observe that the fairness index stays around the 0.91 level.

VII. Practical Considerations

In Sections V and VI, we assumed the use of generalized processor sharing (GPS) model by the session peers to deliver

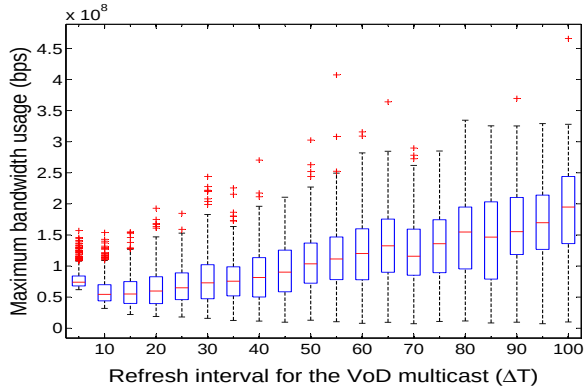


Fig. 26: Maximum servicing overhead for C-WFQ with $mmpp_2$-based arrivals.

the requested content to end users. However, from a practical standpoint, we can only support the sharing of resources at the packet level. In a time-constrained on-demand delivery system, such restrictions on peer delivery process play a significant role on the user perceived performance. Specifically, since resources are allocated at the packet level, latency requirements dictate a lowerbound on each peer's transmission rate. Since the decoding process can only initiate after all the dependent packets are received, the latency for a block of dependent packets is determined by the delivery time of the latest delivered packet within the given block.

Therefore, to satisfy the latency requirements within a packet-by-packet GPS framework [17], we need to make a few modifications to our original architecture. We can summarize the proposed changes as follows:

- First, we use a weighted round robin (WRR) approach to distribute the resources to the set of active transmitters. We assign the weights based on the allocated transmission rates. At each round, session peer ν_i delivers k_i packets, where k_i is determined as follows:

$$k_i = \lfloor W_u / W_{u,\min} \rfloor \tag{18}$$

  where $W_{u,\min}$ represents the minimum uplink bandwidth availability for the given transmitter set. The information on which peer delivers which set of packets is determined by the server and delivered to the active peers over a multicast control channel. Since the delivery rates are only updated after each join/leave event (as in the case of C-WFQ), the overhead associated with the delivery of these packets will be negligible.
- Secondly, we set an upper bound on the number of sessions to which a session peer can contribute. As the number of sessions that a peer services increases, we observe a similar amount of decrease in bandwidth availability per session at the given peer. Because of the latency constraints, we cannot allow a continual decrease in the available bandwidth at the session peers per request. For that purpose, we set a constraint on the system latency and refer to it as $T_{L,sys}$, which represents the maximum delivery latency for a given packet. [9] We can then determine the constraints on the minimum bandwidth availability using the following equation:

$$W^*_{u,\min} = l_p / T_{L,sys} \tag{19}$$

  where l_p represents the packet size. In short, each session peer needs to have a bandwidth availability of at least $W^*_{u,\min}$ to service a request. Any resource allocation rate lower than $W^*_{u,\min}$ results in the request getting dropped and the unused portion of peer resources are distributed to the remaining active requests.

Since the proposed changes only affect how the resources are distributed to each active request and the overall bandwidth usage through the session peers stay constant, we expect the proposed modifications to have a limited impact on the system performance.

In a peer-assisted system, to achieve the desired quality of experience levels we also need to have an explicit error recovery scheme. The proposed framework uses the dedicated server for that purpose. Note that, we are essentially interested in providing reliable delivery during the synchronization phase. We achieve this by dividing the on-demand content into multiple transmission blocks, create repair packets for each block, and deliver them to the clients. The protection levels can be negotiated during the session join phase and, if necessary, the assigned protection levels can be modified based on feedback received from the clients.

Additionally, to satisfy the basic requirements for an on-demand system within the proposed framework (such as *skip* or *fast-forward*), we can assign the user to an ongoing session that was initiated earlier than the most recently created session and deliver the requested packets accordingly, through the session peers and/or the dedicated server. Since the timing information is readily available at the dedicated server, it can make the best (*i.e.*, resource optimal) decision on which session to assign the client to.

Another important concern in our implementations is the processing and communication overheads associated with the update process. In practice, we can alleviate such concerns by dynamically switching between the computationally efficient C-FCFS technique and the resource efficient C-WFQ technique. The characteristics of the request arrival process has been studied extensively over the years (*e.g.*, see [4]) and it has been observed that the incoming load to the system can be estimated in time, which is especially true for the on-demand traffic as it experiences less variations when compared to the request arrival process for the live content. Hence, we can set a threshold for the system load to determine which approach to choose while initiating the sessions to achieve the optimal tradeoffs between processing overhead and resource usage efficiency.

Lastly, we discuss the impact of non-homogeneous bandwidth availability at the client side with specific emphasis on the uplink bandwidth availability due to its enormous impact on peer usage efficiency. In our earlier analysis, we assumed

[9] For the sake of simplicity, we ignore the impact of propagation delay (d_p) on the system latency (D_S), since we assume $D_S \gg E[d_p]$.

the peers to have the same limitations in their capabilities (*i.e.*, sharing the same values for the downlink/uplink bandwidth availability), even though the proposed cooperative delivery techniques explicitly support the use of variable rates. To study the impact of variable rates on the servicing overhead and the fairness performances, we focus on two specific scenarios: (i) constant uplink bandwidth availability (CuBW) scenario which assumes $W_u^{(\nu)} = 1Mbps, \forall \nu \in S$, and (ii) variable uplink bandwidth availability (VuBW) scenario which assumes $W_u^{(\nu)} = Uniform(500Kbps, 1.5Mbps)$ with $E[W_u^{(\nu)}] = 1Mbps$. We illustrate the comparative results for the mean servicing overhead and the fairness performances in Figures 27 and 28.

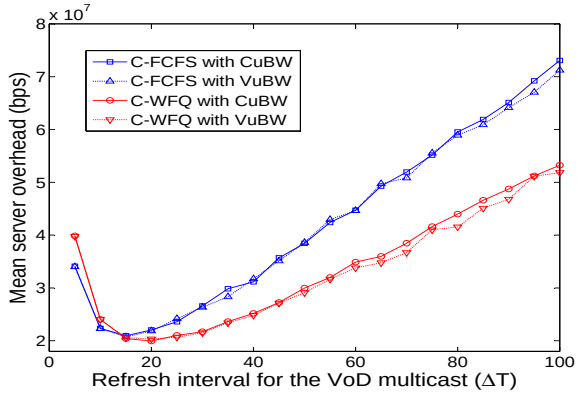


Fig. 27: Mean servicing overhead for variable peer uplink bandwidth availability with HPP-based arrivals.

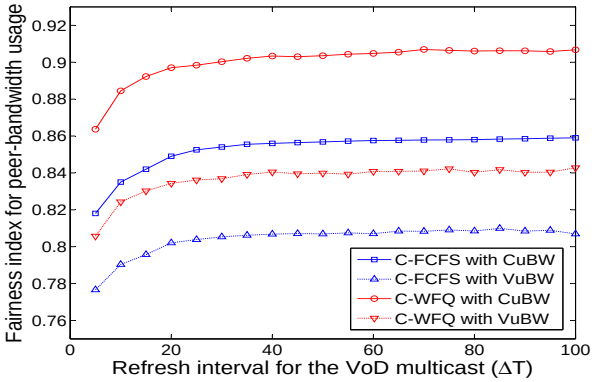


Fig. 28: User fairness for variable peer uplink bandwidth availability with HPP-based arrivals.

Our results suggest that varying the uplink bandwidth availability at the client side has limited impact on the servicing overhead, which is expected as the average peer bandwidth availability converges to the mean value as more peers join the session. Note that, from the perspective of a newly arriving client ν_i, it makes little difference whether or not the earlier arrived peers experience different uplink bandwidth availabilities as the main attention is given on the overall bandwidth availability (*i.e.*, $\left(\sum_{j<i} W_u^{(\nu_j)}\right)$). On the other hand, since each peer contributes to cooperative delivery phase at a different rate (depending on the uplink bandwidth availability), we observe $\approx 6-8\%$ loss in the fairness performance. Also note that, the performance degradations observed in the fairness measure can be reduced by further lowering the maximum allowed peer contributions per session per peer, however, at the cost of increased servicing overhead at the server side.

VIII. Related Work

Multimedia streaming has been studied extensively over the last two decades, leading to many successful implementations over the years (for more detailed overview of the state-of-the-art techniques see [18], [19], [13], [20] and the references within). We can generally categorize these approaches into three main categories: *server-based* solutions, *peer-based* solutions, and *hybrid* solutions (*i.e.*, peer-assisted server-based solutions or server-assisted peer-based solutions).

The earliest content delivery solutions focused on the use of dedicated servers to deliver the requested content using *unicast streams*. Specifically, for each received request, a unicast connection is established between the server and the client over which the video stream is delivered, thereby suggesting a linear increase in the servicing overhead as the number of requests increases. As a result, unicast-based solutions are considered to be impractical due to bandwidth limitations and scalability related concerns. To overcome the limitations introduced by unicast-based solutions, *batching* techniques were proposed that utilize the idea of periodically broadcasting the session streams (*e.g.*, [21], [22], [23]). [10] Within this framework, two approaches have gained the most attention. The first approach assumes that the server broadcasts the whole content in a single session through successively created broadcast streams (*e.g.*, [24]), while the second approach divides the content to multiple segments before delivering them over parallel streams (*e.g.*, [25], [26], [27]). However, regardless of the approach being used, limiting the delivery of session streams to periodic broadcasts typically introduces additional latencies, thereby creating a *near VoD* solution instead. To more effectively utilize the idea of bandwidth-efficient batching in a more time-sensitive manner, *patching* techniques were proposed that combine regular broadcasts (or multicasts) with patching streams (e.g., [28], [29]). Within this framework, a client can join an ongoing multicast session immediately without waiting for the start of the next multicast session, while at the same time receiving the earlier delivered data over a second stream. Then the client can start the decoding process as soon as the buffering requirements are met based on the rate that is used to deliver the patching stream. For instance, in [30] the authors proposed a threshold-based multicast approach that is capable of adapting the frequency of initiating a new multicast session based on the arrival times of requests. In doing so, when compared to policies that initiate non-overlapping multicast sessions, bandwidth requirements can be significantly reduced. These improvements can be further increased by introducing multicast-based patching

[10] Note that, in earlier works, the term *broadcast stream* is used to refer to streams that are generated with no request from the client side, whereas the term *multicast stream* is used to refer to streams that are generated based on client requests. In the remainder of this section, we use these terms interchangeably as we essentially focus on the delivery process.

streams, and allowing clients to merge using two or more multicast streams (*e.g.*, [31]).

However, despite these advantages, server-based solutions typically underperform when servicing a large set of clients. Therefore, to achieve the desired performance results, we need to either limit the number of clients (*e.g.*, by not admitting new users to the systems or increasing the blocking probability) or increase the number of servers (which may significantly increase the operational costs). Neither result is desired, and so to achieve a more scalable and dynamic delivery framework, peer-based solutions were proposed. These solutions initially focused on the delivery of live streams to limit the overhead at the client side [32], [33]. However, due to shifting user interests from live to on-demand streaming services, resource limitations at the peers have started to become a major limiting factor in achieving the targeted quality of experience (QoE) levels [34]. The unique characteristics of the on-demand content delivery services (*e.g.*, users receiving the same content at different timings or supporting functionalities like fast-forward/rewind) was a major factor for the perceived limitations and the efficiency loss observed in peer-based solutions.

To solve the limited-resources problem in a more scalable framework, cooperation among peers is integrated as a feature to support server-based solutions by requiring peers to implement a distributed caching system (see [35], [36] for earlier discussions). Specifically, each peer is required to allocate a small amount of their local storage (*e.g.*, a few hundred MBs to a few GBs) to support this process. By distributing resources to session peers and using them as a potential delivery source, we can significantly reduce the server-side requirements [37]. Our approach falls in this category.

Over the last decade, various hybrid-based solutions were proposed, and these approaches typically differ in the ways the content is segmented, the segments are distributed to peers, the delivery of segments are scheduled at the peers, and the server is utilized (*e.g.*, [38], [39], [40], [41], [42], [16]). One of the earliest solutions in this category was CoopNet, which only utilized the peers, who have previously cached the content they had received, during server overloads [38]. To minimize the problems associated with node/link failures and to improve the average servicing quality, multiple description coding (MDC) is used on the content delivered through the peers. However, since the peers are utilized only during overload periods, flash crowds can easily lead to dropped requests and additional processing delays. In [39] the authors proposed a tree-based approach, referred to as *P2Cast*, that is based on the idea of grouping requests, which arrive to system close in time, to form a session by creating a multicast tree at the application layer, which is referred to as the base tree. Clients then receive the session data by first joining the base tree, which is initially created by the server, and then receiving the patching service through the server or an earlier-arrived peer. However, patching service is limited to a single source and the threshold parameter, that is used to decide on which requests to group, is not updated based on the incoming load. In [40] the authors proposed a batching-based framework, which utilized a chaining-based patching service. Specifically, session multicasts are initiated at the server at specific intervals and, upon joining a session, a client ν_i receives two streams, session multicast from the server and patching data from the previous client, ν_{i-1}. However, since the multicast sessions are streamed over non-overlapping intervals, late arriving peers can put significant burden on the system, thereby leading to poor fairness results. A similar approach was proposed in [42], referred to as the P2P batching (PPB) approach, which allowed the patching process to use a single peer as a source. However, the success of the PPB approach relies on the assumption that each peer has an uplink bandwidth availability that equals the delivery rate for the session multicast (W_M), which may not always be the case, thereby limiting the use of peers for the patching service. Furthermore, the problems associated with time-varying arrival rates and the resultant impact on the proposed policy, and the fairness related concerns are not investigated. In [41] a push-based peer-delivery model is proposed, which is based on the idea of *proactive resource caching*. Specifically, peers are expected to acquire the on-demand content proactively from the content delivery server during low activity periods and contribute to the resource distribution phase using the locally stored content. However, in a diverse network that consists of unwilling peers or peers with limited resources a push-based model may not be a feasible solution.

The closest approach to ours to date was proposed in [16], and it was referred to as the cooperative peer-assist and multicast (CPM) approach. The framework proposed in [16] separates the video content into 30-second long chunks and each chunk is delivered separately by either the assisting peers or the dedicated video server. To receive these chunks, clients need to sequentially go through the video server, the directory server, and the peers. If the request can be serviced by the video server, using an already scheduled multicast before the deadline associated with the chunk expires, then the request is serviced through the video server. Otherwise, peers are probed to deliver the requested chunk. However, this probing process may introduce additional latencies and communication overhead. Furthermore, even though the authors showed significant improvements in the servicing overhead, relying on a single peer to deliver a long continual chunk is not a practical approach unless it is supported by explicit error recovery techniques, which may also lead to inefficient use of system resources. To improve the recovery performance during such phases, in CPM, clients are allowed to switch to a different peer to receive the requested chunks at the cost of additional latency. Also note that, in the case of local failures, CPM cannot guarantee the reliable delivery of the requested session data to the client.

Unlike the CPM approach, our framework supports communication in a more fluid way, as the server is responsible for assigning the peers and initiating the delivery process at the time of request and at each point of system state change (*i.e.*, incoming peers or start of a new service, or, departing peers or finish of an earlier service). As the clients request resources from the peers that have immediate access to the requested content through the same multicast, no additional search is required to find the cooperatively delivering peers.

Furthermore, we explicitly address the fairness concerns in our framework by putting strict limits on peer contributions, whereas in CPM fairness is achieved implicitly during the initial peer assignment phase using a randomized selection without taking into account other measures. Another crucial difference between CPM and our approach is observed in the use of chunks for delivering the session data. Because of the messaging overhead CPM introduces for each chunk, smaller sized chunks cannot be utilized efficiently, whereas our work can dynamically adjust the chunk size depending on the number of peers connected to a session without introducing a significant overhead (as update messages are only transmitted during peer-join or service-finish instances).

IX. Conclusion

In this paper, we developed two novel cooperative delivery techniques with the objective of improving the scalability performance of the network by minimizing the reliance on the use of video server for the delivery of on-demand content. For that purpose, we used the session peers to distribute the resources in the network. Additionally, we integrated fair resource allocation policies into the proposed content delivery techniques to maximize the cooperation among session peers. We dynamically adapt the peer delivery rates and delivery size for requests based on the system state. We performed extensive simulation studies to analyze the most critical performance measures for the proposed framework and showed the advantages of using the developed techniques. We observed significant improvements in the server utilization rate allowing the network to scale more efficiently in the number of clients supported and the number of services provided.

References

[1] W.-Pk. Yiu, Xing Jin, and S.-H.G. Chan, "Challenges and approaches in large-scale P2P media streaming," *IEEE Multimedia*, vol. 14, no. 2, pp. 50–59, 2007.

[2] P. Gill, M. F. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *ACM IMC*, 2007, pp. 15–28.

[3] Y. Lu, B. Fallica, F. A. Kuipers, R. E. Kooij, and P. V. Mieghem, "Assessing the quality of experience of SopCast," *International Journal of Internet Protocol Technology*, vol. 4, no. 1, pp. 11–23, 2009.

[4] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A measurement study of a large-scale P2P IPTV system," *IEEE Transactions on Multimedia*, vol. 9, no. 8, pp. 1672–1687, Dec 2007.

[5] Y. Xiao, X. Du, J. Zhang, F. Hu, and S. Guizani, "Internet protocol television (IPTV): The killer application for the next-generation Internet," *IEEE Communications Magazine*, vol. 45, no. 11, pp. 126–134, Nov 2007.

[6] I. Bermudez, M. Mellia, and M. Meo, "Investigating overlay topologies and dynamics of P2P-TV systems: The case of SopCast," *IEEE JSAC*, vol. 29, no. 9, pp. 1863–1871, Oct 2011.

[7] K. Kerpez, D. Waring, G. Lapiotis, J.B. Lyles, and R. Vaidyanathan, "IPTV service assurance," *IEEE Communications Magazine*, vol. 44, no. 9, pp. 166–172, 2006.

[8] P. Diminico, V. Gopalakrishnan, R. Jana, K.K. Ramakrishnan, D.F. Swayne, and V.A. Vaishampayan, "Capacity requirements for on-demand IPTV services," in *3rd International Conference on Communication Systems and Networks (COMSNETS)*, Jan 2011, pp. 1–10.

[9] C. Costa, I. Cunha, A. Borges, C. Ramos, M. Rocha, J. Almeida, and B. Ribeiro-Neto, "Analyzing client interactivity in streaming media," in *ACM WWW*, 2004, pp. 534–543.

[10] M. Vilas, X.G. Paneda, R. Garcia, D. Melendi, and V.G. Garcia, "User behavior analysis of a video-on-demand service with a wide variety of subjects and lengths," in *31st EUROMICRO Conference on Software Engineering and Advanced Applications*, 2005, pp. 330–337.

[11] W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," in *Proceedings of ACM EuroSys*, 2006, pp. 333–344.

[12] T. Qiu, Z. Ge, S. Lee, J. Wang, Q. Zhao, and J. Xu, "Modeling channel popularity dynamics in a large IPTV system," in *ACM SIGMETRICS/Performance'09*, 2009, pp. 275–286.

[13] K. A. Hua, M. A. Tantaoui, and W. Tavanapong, "Video delivery technologies for large-scale deployment of multimedia applications," *Proceedings of the IEEE*, vol. 92, no. 9, pp. 1439–1451, Sep. 2004.

[14] A. Sendonaris, E. Erkip, and B. Aazhang, "User cooperative diversity," *IEEE Tr. on Communications*, vol. 51, no. 11, pp. 1927–1948, Nov 2003.

[15] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," DEC Research Report TR-301, Sep 1984.

[16] V. Gopalakrishnan, B. Bhattacharjee, K.K. Ramakrishnan, R. Jana, and D. Srivastava, "CPM: Adaptive video-on-demand with cooperative peer assists and multicast," in *IEEE INFOCOM*, 2009, pp. 91–99.

[17] A. Parekh, *A Generalized Processor Sharing Approach to Flow Control*, Ph.D. thesis, MIT, 1992.

[18] L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popular videos," in *ACM NOSSDAV*, 1998.

[19] A. Hu, "Video-on-demand broadcasting protocols: A comprehensive study," in *IEEE INFOCOM*, 2001, pp. 508–517.

[20] J. Choi, A. Reaz, and B. Mukherjee, "A survey of user behavior in vod service and bandwidth-saving multicast streaming schemes," *IEEE Communications Surveys and Tutorials*, vol. 14, no. 1, 2012.

[21] C. C. Aggarwal and J. L. Wolf, "On optimal batching policies for video-on-demand storage servers," in *IEEE International Conference on Multimedia Computing and Systems*, 1996, pp. 253–258.

[22] H. J. Kim and Y. Zhu, "Channel allocation problem in VoD system using both batching and adaptive piggybacking," *IEEE Transactions on Consumer Electronics*, vol. 44, no. 3, pp. 969–976, Aug. 1998.

[23] N. L. S. da Fonseca and R. D. A. Facanha, "The look-ahead-maximize-batch batching policy," *IEEE Tr. on Multimedia*, vol. 4, no. 1, pp. 114–120, Mar. 2002.

[24] K. C. Almeroth and M. H. Ammar, "The use of multicast delivery to provide a scalable and interactive video-on-demand service," *IEEE JSAC*, vol. 14, no. 5, pp. 1110–1122, Aug. 1996.

[25] S. Viswanathan and T. Imielinski, "Pyramid broadcasting for video on demand service," in *IEEE MMCN*, 1995, pp. 66–77.

[26] K. A. Hua and S. Sheu, "Skyscrapper broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems," in *ACM SIGCOMM*, 1997, pp. 89–100.

[27] L. Juhn and L. Tseng, "Fast data broadcasting and receiving scheme for popular video service," *IEEE Tr. on Broadcasting*, vol. 44, no. 1, pp. 100–105, Mar. 1998.

[28] K. A. Hua, Y. Cai, and S. Sheu, "Patching: a multicast technique for true video-on-demand services," in *ACM ICM*, 1998, pp. 191–200.

[29] Y. Cai, K. A. Hua, and K. Vu, "Optimizing patching performance," in *IS&T/SPIE MMCN*, 1999, pp. 204–215.

[30] L. Gao and D. Towsley, "Threshold-based multicast for continuous media delivery," *IEEE Tr. on Multimedia*, vol. 3, no. 4, pp. 405–414, Dec. 2001.

[31] D. Eager, M. Vernon, and J. Zahorjan, "Minimizing bandwidth requirements for on-demand data delivery," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 5, pp. 742–757, 2001.

[32] X. Zhang, J. Liu, B. Li, and T. S. P. Yum, "Coolstreaming/donet: A data-driven overlay network for efficient live media streaming," in *IEEE INFOCOM*, 2005, pp. 2102–2111.

[33] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *IEEE INFOCOM*, 2006, pp. 1–10.

[34] Y. Huang, T. Z. J. Fu, D.-M. Chiu, J. C. S. Lui, and C. Huang, "Challenges, design and analysis of a large-scale P2P-VoD system," in *ACM SIGCOMM*, 2008, pp. 375–388.

[35] S. Acharya and B. C. Smith, "MiddleMan: A video caching proxy server," in *ACM NOSSDAV*, 2000.

[36] S. Paknikar, M. Kankanhalli, K. R. Ramakrishnan, and S. H. Srinivasan, "A caching and streaming framework for multimedia," in *ACM Multimedia*, 2000, pp. 13–20.

[37] C. Huang, J. Li, and K. W. Ross, "Can Internet video-on-demand be profitable?," in *ACM SIGCOMM*, 2007, pp. 133–144.

[38] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *ACM NOSSDAV*, 2002.

[39] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2cast: Peer-to-peer patching scheme for VoD service," in *ACM WWW*, 2003, pp. 301–309.

[40] J.-F. Paris, "A cooperative distribution protocol for video-on-demand," in *IEEE ENC'05*, 2005, pp. 240–247.

[41] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Varvello, "Push-to-peer video-on-demand system: Design and evaluation," *IEEE JSAC*, vol. 25, no. 9, pp. 1706–1716, Dec. 2007.

[42] K.-M. Ho, W.-F. Poon, and K.-T. Lo, "Video-on-demand systems with cooperative clients in multicast environment," *IEEE Tr. on Circuits and Systems for Video Technology*, vol. 19, no. 3, pp. 361–373, Mar. 2009.

APPENDIX
EXPECTED BANDWIDTH USAGE AT THE SERVER SIDE

We start our analysis by initially making the following assumptions:

‡ To simplify our analysis, we assign integer values to ρ, *i.e.*, $\rho \in \mathbb{Z}$.

‡ Session join requests arrive in intervals of μ seconds, where $\mu = 1/\lambda$ (since the arrival process is based on the Poisson process). [11]

‡ System time is initialized to $\tau = 0$.

We determine the time-averaged servicing rate by evaluating the resulting servicing overhead for each newly arrived request.

◇ CASE I: $\rho = 2$

For the first case, we assume $W_M = 2W_u$, where W_u represents the uplink bandwidth at the user side. The first request, ι_0 made by ν_0, initiates the multicast session and requires no additional unicast streaming from the server. The second request, ι_1 made by ν_1, arrives at μ and can be serviced by two sources, server and ν_0. Since ν_0 can service at a rate of W_u, to satisfy the following servicing requirements:

$$\frac{\sum_{i \le l} \left(W_{rcv,i} \times \delta t_i\right)}{\sum_{i \le l} \delta t_i} \ge W_M, \tag{20}$$

remaining resources are received from the server at a rate of W_u.

Since ι_1 is serviced at W_M, its servicing finishes at $\tau_2 = 2\mu$. Hence, the third request (ι_2 by ν_2) that arrives at 2μ can immediately start receiving synchronization data from both ν_0 and ν_1. As a result ι_2 requires no additional resources from the server. Since ι_2 is serviced at W_M, its servicing finishes at 4μ.

When the fourth request (ι_3 by ν_3) arrives to the system at $\tau_3 = 3\mu$, it initially sees only ν_2 as available, a condition that stays valid till $\tau_4 = 4\mu$. As a result, until τ_4, ν_3 is also serviced by the server at a rate of W_u. During the interval (τ_3, τ_4), ν_3 receives one third of its requested data. After t_4, it can receive the remaining data ($2W_M\mu$) from all the available peers, for a total rate of $3W_u = (3/2)W_M$, leading to a finish time of $4\mu + 4\mu/3$. We can recursively use this procedure to determine the resources required at the the server. We show some of these results in Table II, where ι represents the request count, τ_A (or τ_F) represents the request arrival (or finish) time, and $B^{(\rho)}_{S,u}(\iota)$ represents the amount of data delivered by the server to service the ιth request.

[11] In the text, we explained how a multicast session is created, at ΔT intervals, to accommodate the requests received during the next ΔT period. In practice, the optimal solution necessitates a new multicast session to be created only after receiving a request for it. Hence, often a non-zero gap exists between successive sessions. However, the 2nd assumption allows the sessions to be created at the boundary instances so long as ΔT is an integer multiple of μ.

TABLE II: $\rho = 2$

ι	τ_A	$B^{(2)}_{S,u}(\iota)$	τ_F
1	μ	μW_u	2μ
2	2μ	0	4μ
3	3μ	μW_u	$4\mu + 4\mu/3$
4	4μ	$4\mu W_u/3$	$4\mu + 4\mu/3 + 4\mu/3$
5	5μ	$5\mu W_u/3$	$4\mu + 8\mu/3 + 4\mu/3$
...	...	...	...
$k > W_M/W_u$	$k\mu$	$k\mu W_u/3$	$4\mu + 4(k-2)\mu/3$

If we sum up the servicing bandwidth requirements for requests up to the Nth, we obtain the following equation for the unicast servicing bandwidth:

$$B^{(2)}_{S,u} = \sum_{\forall \iota} B^{(2)}_{S,u}(\iota) = \frac{W_M \times \mu \times (N^2 + N)}{12} \tag{21}$$

≀◇

◇ CASE II: $\rho = 3$

For the second case, we set $W_M = 3W_u$. The only difference between the current case and the previous is when the peers start receiving at a rate greater than the source multicast rate W_M, which occurs after the fourth received request. The results for the second case are shown in Table III.

TABLE III: $\rho = 3$

ι	τ_A	$B^{(3)}_{S,u}(\iota)$	τ_F
1	μ	$2\mu W_u$	2μ
2	2μ	$2\mu W_u$	4μ
3	3μ	$2\mu W_u$	6μ
4	4μ	$4\mu W_u$	$15\mu/2$
5	5μ	$5\mu W_u$	9μ
...	...	...	...
$k \ge W_M/W_u$	$k\mu$	$k\mu W_u$	$3(k+1)\mu/2$

Similar to the previous case, we can determine the bandwidth requirements for the unicast service as follows:

$$B^{(3)}_{S,u} = \sum_{\forall \iota} B^{(3)}_{S,u}(\iota) = \frac{W_M \times \mu \times (N^2 + N)}{6} \tag{22}$$

≀◇

We obtained similar expressions for the other scenarios as well, allowing us to generalize the bandwidth requirements as follows:

$$E[B^{(\rho)}_{S,u}] = \frac{W_M \times \mu \times (N^2 + N)}{2 \times (\rho^2 + \rho)/(\rho - 1)^2} \tag{23}$$

We can determine the overall bandwidth requirements at the server side by summing up the unicast and multicast bandwidth requirements. For that purpose, we used a session-based approach, *i.e.*, we focused on the lifetime of a single session. Assuming that the multicast delivery for the originating session starts at $\tau = 0$, we focused on the requests that arrive during the lifetime of that session, i.e., requests arrive within $(0, L_S)$. For the given approach, we can determine the overall bandwidth requirements as follows:

$$B^{(\rho)}_S = E[B^{(\rho)}_{S,u}] \times \left(\kappa - 1 + \varsigma/\Delta T\right) + \kappa W_M L_S \tag{24}$$

where κ equals the number of multicast sessions created during the lifetime of a single session and is given by $\lceil L_S/\Delta T \rceil$, and

ς equals $L_S-(\kappa-1)\Delta T$. The activity period for the κ sessions depends on the speed of processing for the received requests and it equals $L_S + (\kappa - 1)\Delta T$ in the worst case (*i.e.*, from the perspective of *maximum bandwidth utilization*, worst case scenario occurs when the servicing of the requests finishes by the end of the last session multicast) or $2L_S$ in the best case (best case scenario occurs when the last arrived request, which arrives at $\approx L_S$ is serviced on average at the source multicast rate). Using the worst case scenario, we can determine the average servicing overhead as follows:

$$E[W_S^{(\rho)}] = \frac{B_S^{(\rho)}}{L_S + (\kappa - 1)\Delta T} \tag{25}$$

In the text, we used an approximation on the above approach, by setting κ to L_S/Δ and approximated the duration of the activity period as $2L_S$ independent of the value assigned to ΔT. Note that the approximation yields a root mean squared error value of $\approx$ 0.94Mbps for our baseline example (i.e., W_M = 3Mbps, μ = 20ms, L_S = 100m, and ρ = 3), when compared to the results for the worst case scenario.

We used a similar procedure for the C-WFQ case. The only difference between the two scenarios is that, for C-WFQ, we assumed the client to divide its uplink bandwidth equally to all the requests that it had received and accepted.